Social Reasoning in Multi-Agent Systems with the Expectation-Strategy-Behaviour Framework

Iain Andrew Wallace

Doctor of Philosophy

Centre for Intelligent Systems and their Applications
School of Informatics
University of Edinburgh

2010

Abstract

Multi-agent Systems (MAS) provide an increasingly relevant field of research due to their many applications to modelling real world situations where the behaviour of many individual, self-motivated, agents must be reasoned about and controlled. The problem of agent social reasoning is central to MAS, where an agent reasons about its actions and interactions with other agents. This is the most important component of MAS, as it is the interactions, cooperation and competition between agents that make MAS a powerful approach suited for tackling many complex problems. Existing work focuses either on specific types of social reasoning or general purpose agent practical reasoning - reasoning directed toward actions. This thesis argues that social reasoning should be considered separately from practical reasoning. There are many possible benefits to this separation compared to existing approaches. Principally, it can allow general algorithms for agent implementation, analysis and bounded reasoning. This viewpoint is motivated by the desire to implement social reasoning agents and allow for a more general theory of social reasoning in agents. This thesis presents the novel Expectation-Strategy-Behaviour (ESB) framework for social reasoning, which provides a generic way to specify and execute agent reasoning approaches. ESB is a powerful tool, allowing an agent designer to write expressive social reasoning specifications and have a computational model generated automatically. Through a formalism and description of an implemented reasoner based on this theory it is shown that it is possible and beneficial to implement a social reasoning engine as a complementary component to practical reasoning. By using ESB to specify, and then implement, existing social reasoning schemes for joint commitment and normative reasoning, the framework is shown to be a suitable general reasoner. Examples are provided of how reasoning can be bounded in an ESB agent and the mechanism to allow analysis of agent designs is discussed. Finally, there is discussion on the merits of the ESB solution and possible future work.

Acknowledgements

Many thanks to my supervisor, Michael Rovatsos, for his ideas, guidance and support throughout the development of this thesis. Thanks also to my second supervisor, Alan Smaill, for his guidance and comments. I would also like to thank my fellow students and researchers in CISA for sharing and listening to ideas. In particular I would like to thank members of the Agents Group; Alexandros Belesiotis, George Christelis, Matt Crosby and Francesco Figari. Their help with ideas and discussion whenever the Science-o-Meter swung into the red was invaluable. Thank you also to my family, for their support throughout.

Finally, many thanks to Ruth for her endless support and encouragement.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Iain Andrew Wallace*)

Table of Contents

List of Figures

Chapter 1

Introduction

Multi-agent Systems (MAS) provide an increasingly relevant field of research due to their many applications to modelling real world situations where the behaviour of many individual, self-motivated, "agents", autonomous AI or otherwise, must be reasoned about or controlled. Examples of MAS include scenarios such as online trading e.g. eBay, cooperating flocks of unmanned aerial vehicles (UAVs), teams of football playing robots and grid computing resources negotiating to provide a service.

This thesis is concerned with "social reasoning". This refers to the influences on an agent when deliberating interactions with other agents. This is a subset of practical reasoning, which is concerned with how an agent chooses an action based on its view of the environment. Social reasoning theories, whilst studied extensively in isolation, are hard to implement directly in agents. They often only specify properties of beliefs or behaviours but not the way these should affect the computational reasoning mechanisms of a concrete agent design.

This thesis argues that social reasoning should be considered separately from practical reasoning concerns. This viewpoint is motivated by the desire to implement socially reasoning agents and allow for a more general theory of social reasoning in agents. Therefore the objective was to create a general social reasoning framework, with both theory and implementation. This framework can be evaluated by considering its suitability to capture common social reasoning concepts and execute them.

A formal theory and implementation of the Expectation-Strategy-Behaviour (ESB) framework is presented, which is an extension to a practical reasoner. Through implementations of several reasoning schemes and demonstrations of generally applicable

simplification algorithms that ESB makes possible, the advantages of considering social reasoning in this way are illustrated. The overall contribution of this work is an extension of general practical reasoning systems and can also be seen as a generalisation of the wide variety of work on specific social reasoning problems.

1.1 Social Reasoning and Multi-Agent Systems

Multi-agent systems have a lot in common with traditional distributed computing but present several unique challenges: agents are not necessarily cooperative, homogenous nor the population static. In addition distributed systems are specified then designed and their behaviour could be well known, whereas an open MAS may change and evolve as the agent population changes. The possibility for a changing population is one problem area for social agents. If others may enter and leave the system at will, is there any value in modelling the behaviour of individuals or is it more efficient to simply model classes of interactions? The possibility of uncooperative agents raises the issue of trust and the related concepts of virtual organisations and other methods for grouping trusted individuals.

Systems of social norms are often proposed as a way to structure the system and allow for easier interactions. Indeed there are plenty of examples of norms in the real world and in domains in which agents may wish to operate. Norms differ from the rules of a system in that they can change and may be broken. It is necessary for an agent to consider the implications of this, as it may wish to disobey norms to increase its short or long term utility. In closed MAS systems, where the specification of all agents may be known at design time, there are still many challenges. Agents may be heterogeneous in operation or capability, communications may be unreliable or agents may fail to complete tasks allocated to them. Finally, the system may be simply too large and complex to consider all possible interactions during design. As well as the difficulty of designing agents capable of useful behaviour in these situations there is also the issue of system specification. Although agent systems may be autonomous, many use cases have them ultimately acting on someone's behalf, with some level of authority. Accurate specification and being able to at least place bounds on their possible behaviour may be necessary for a user to trust them.

This wide set of challenges forms the area of *social reasoning*. The area is introduced

briefly now, as the definition of social reasoning and the class of problems it covers is core to the work in this thesis. The next chapter provides a background and further expansion on the approaches making up this area by way of a survey of the state of the art in agent social reasoning. This also serves to situate the work in the field and provide the motivation and inspiration for the problem tackled by this thesis.

1.2 Problem Definition

Before diving into the area in more detail, it is instructive to introduce the chosen problem and sketch the outline of the resulting solution. The main conclusions that are presented in Chapter 2 can be summarised as follows:

1. In open MAS, social reasoning is defined by the need for dynamic belief update, but most current approaches do not address this directly in the reasoning architecture. The social reasoning is "designed in". A general framework for handling reasoning about change in the social context could provide many benefits. Different approaches to reasoning could be combined, algorithms to simplify and bound reasoning could be generally applied and it could guide implementation.

2. Many approaches focus only on one facet of the social reasoning problem, ignoring other aspects, and these different approaches may be at differing levels of abstraction. A unifying framework to describe, compare and reason about these approaches would be useful for further research in this area.

3. The relationship between practical and social reasoning is important and there may be benefits to considering social reasoning as a separate, though linked, process to practical reasoning. Defining social reasoning and the relation to practical reasoning more precisely may enable the design of simpler or more powerful social agents.

The proposed solution to these problems is the Expectation-Strategy-Behaviour (ESB) framework, that provides a logical theory, method and implementation by considering social reasoning at the general level as discrete from practical reasoning. The description of the problems tackled in this thesis can be split up into the hypothesis driving the work, the motivation behind it, and the specific goals targeted.

1.2.1 Hypothesis

The hypothesis driving this work derives from the above observations and concludes that a general framework for handling reasoning about change in the social context could provide many benefits. Considering this to be the common element in these observations, leads to the following thesis:

> It is possible to separate social reasoning from practical reasoning, allowing for a generic specification of social reasoning schemes. This will allow for the development of generic algorithms for bounded execution of agent reasoning and analysis of designs.

This is most closely related to observation 3 above – this thesis argues that the separation of social reasoning is necessary to consider it in general terms. Considering observation 1, and dynamic social situations, this is clearly something which any general theory must be capable of. So the framework developed to satisfy the above thesis must also be suitable for the investigation of dynamic scenarios. To develop a complete social reasoning framework, it would seem sensible to first develop the logical foundations and from this base create a reasoning architecture and then implementation. A logical framework will allow for precise specification of properties and provide guidance for an implementation. A general framework such as this is as required for problem 2 above. However even though a logical framework forms a large part of this research, the goal is not comparison and study of existing approaches, but the investigation of the advantages of separating out social reasoning. To actually answer the question, agent designs using this new framework will need to be compared to designs meeting the same overall specification but using existing architectures or logics. An expected consequence of the developed techniques would be that they allow for a simpler design to meet the same specification, or specification of agents with capabilities beyond any single existing approach. By representing different reasoning schemes in a common way their combination may be easier.

The above thesis, although succinct, leaves several details unanswered, which are covered in the rest of this thesis. Firstly, what does it mean to separate social reasoning? Chapters 3 and 4 cover the method proposed to answer this question and present a design of how this separation is achieved. Secondly, how can social reasoning be bounded or analysed in general terms? Section 5.3 covers general algorithms for simplification and bounding reasoning at execution time. Finally, there is the question of why this is an important question to answer and why no acceptable solution currently exists. This

is discussed in the review of the current literature in Chapter 2.

1.2.2 Motivation

To first consider the widest scope, MAS are an important area of research. At present there are only a relatively small number of real-world implementations of systems that could truly be called agents – autonomous, intelligent, self-interested and goal directed. However these still give a flavour of the massive potential for MAS. In the physical world, agents have controlled space probes and unmanned aircraft [NASA, Scerri et al., 2008]. Online agent systems already exist to serve our goals in market trading and with the increase of online interaction throughout society it is easy to imagine agents performing even more tasks for us online e.g. why not an agent to do the shopping? Of course the uses of agents are not just to replace human control. Agent technology has been used to model security at airports, manage power in data centres and model business processes [Kiekintveld et al., 2008, AgentLink]. Computers are ever more present in our lives and ever more connected with the spread of the internet – from universities to businesses, from our business to our homes, from our homes to the devices in our pockets. With all these connected devices and the demand for more intelligence in our devices, so the demand for agent based technologies will grow.

Social reasoning in particular is a crucial field of development. What makes all these example domains fascinating, useful and powerful is not the agents in the individual, but rather how they can work together and interact. The specifics of various forms of interaction are well studied – as will be shown in Chapter 2 – but social reasoning in the general is little considered. This will be a necessary step to combine social reasoning techniques – after all, an agent that reasons about the norms and laws of the system it finds itself in may also be required to form teams, or deceive others to achieve its goals. At the same time, the abstraction of many of these approaches presents a barrier – what good are theories for agent reasoning if we cannot *implement* such agents? Of course they can be applied to designs for agents, but more power, more adaptability, more intelligence can surely be ascribed our agents if they can reason for themselves.

The consideration of implementing social reasoning in a general way carries with it additional motivation. A social reasoning framework could allow for considering agent reasoning in more general terms, for the purpose of analysis, or algorithms to simplify and bound execution or carry out verification of design properties. Describing

reasoning more generally and a generic means for implementation can also aid agent designers, providing an easy route to implementing complex social reasoning schemes, advancing the goal of more intelligent agents.

This completes the motivation only at a general level; the background chapter will reinforce and provide evidence for all the points made here. Given this motivation we now consider the problem in more detail.

1.2.3 Approach

The problem tackled in this thesis can be generally described as the design, implementation and evaluation of a generic social reasoning framework. This can be split into several sub-problems:

Identifying general properties of social reasoning schemes. This is principally carried out through review of the related literature.

Formally describing a social reasoning framework. It is first necessary to specify a framework accurately, to help create it and better analyse its properties. This approach ensures that each aspect of the framework is defined precisely, which aids the implementation of a generic reasoner and its discussion. This task can be broadly split into specifying the semantics of the framework and the workings of the reasoner.

Design of a reasoner. The motivation for this project includes the lack of social reasoning implementations and one of the goals is to create a generic framework that allows for implementation. So at least one implementation is required, which follows the reasoning scheme described formally.

Creation of examples. This is the first part of the evaluation process. Both the theoretical descriptions and implementation will require case studies to illustrate and evaluate them, aid discussion of the merits and weaknesses of the approach taken and allow for evaluating the success with regards to the proposed thesis.

Discussion. With a general framework, it is hard to find metrics that can be applied to measure the success. Instead it is necessary to evaluate analytically and attempt to objectively assess the reasoning system against these stated goals and the validity of the hypothesis.

Solving these problems will lead to the main contributions of the thesis:

- The formal semantics of the ESB framework for social reasoning.
- The algorithms for executing an ESB specification which allows for the creation of agents that use ESB for social reasoning.
- An implementation of an ESB reasoning engine, paired with an existing practical reasoner.
- Algorithms for bounding and analysing agent social reasoning.
- Several example implementations of existing social reasoning schemes in ESB.
- An analysis of the results, as well as the benefits and problems of using the suggested approach.
- More abstract discussion of the merits of general social reasoning, distinct from practical reasoning.

These contributions will be described in detail in the coming chapters.

1.3 An Overview of ESB

To provide a brief overview of the framework developed in this thesis for illustration purposes, a short overview of ESB is sketched out here. This will set the scene for the background chapter, as work can be related to the ESB solution. The ESB architecture is an abstract model for practical social reasoning systems based on the concept of *expectations*. An expectation is a conditional belief regarding a statement whose truth status may be eventually verified by a test and reacted to by the agent holding the expectation. Tests are split into $+$ and $-$ events that confirm or disappoint the expectation. Responses add or remove expectations.

An example based on the card game Rummy [Rummy.com] can be used to illustrate ESB. It is only required to understand that players are trying to collect either runs of cards in a suit, or sets of the same value card. Cards provide a good domain, as an opponent's strategies are known, but defined by the unknown cards they hold. It is an intuitive example of reasoning about another agent. The example models simple reasoning an agent might carry out surrounding the pickup of a $2\diamondsuit$.

Consider agent A expecting agent B to execute some action upon receiving a promise to do so. The expected belief is "B will perform an action (Φ) if he has promised to do so (C)", the test (T) is observing the action or its consequences. The response might be A rewarding B if the promise is kept (ρ^+), or not (ρ^-); or, the expectation itself might be modified, for example altering the level of trust required of B to accept its promises.

The graph in Figure 1.1 below is an "expectation graph" for the Rummy example. The vertices are states containing active expectations and the edges mark the transitions between states specified by the responses (ρ). Strategies constrain the style and scope of the agent's reasoning process and are a restriction of the graph based on the current state. The highlighted portion shown is a strategy applied to state 5 considering states to depth 1.

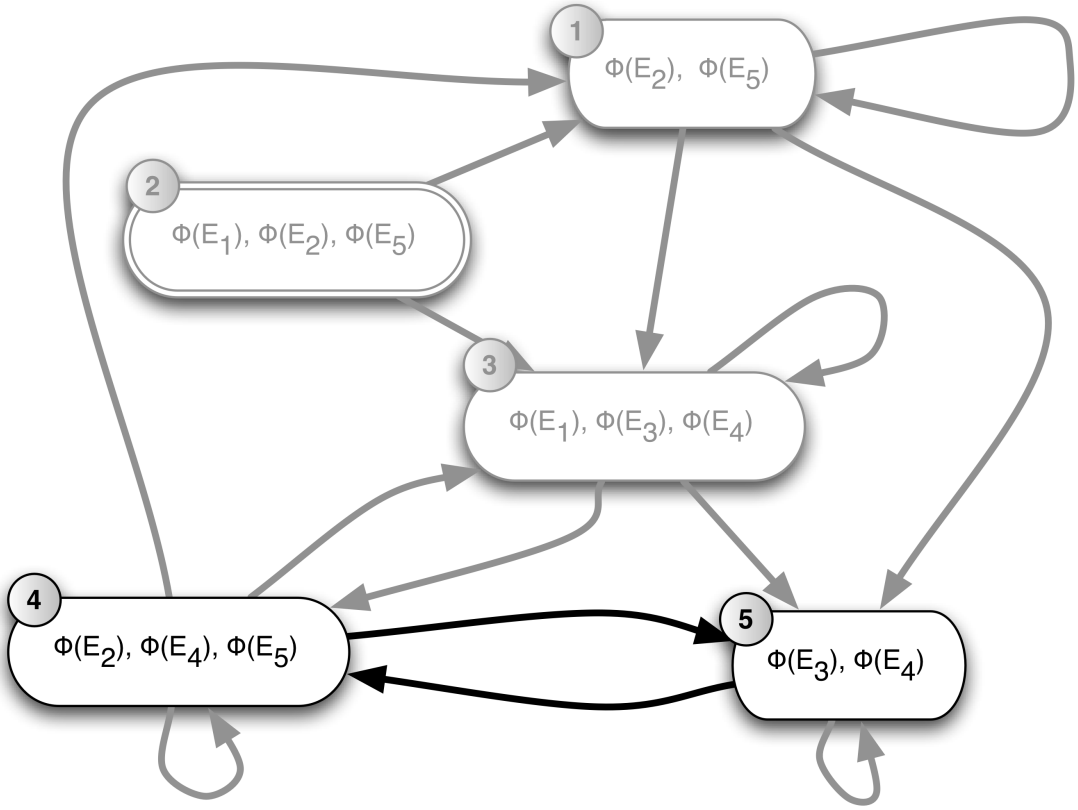

Figure 1.1: An Example ESB Expectation Graph

Behaviours form the link between the representation of social concepts in the expectations and the practical reasoning process. They are actions linked to conditions on the strategy graph. Therefore, conditions are on current expectations and possible future expectations – those in states reachable from the current state according to the strategy graph. For example, in state 5, $\Phi(E_2)$ is a possible future expectation (accessible in

state 4) and could mean "collecting a $\diamondsuit$ run". A behaviour that would hold in this case could be "Do not discard a $\diamondsuit$ if I expect B to be collecting them in the future" and the corresponding action would be to discard a different suit.

This is of course only a most basic description of ESB and more detail can be found later in this document. However it gives the reader a flavour of the sort of solution developed for the problems discussed in the previous section.

1.4 Thesis Structure

The remainder of this document is split into five chapters. These start with the background chapter which provides an overview of social reasoning in MAS, motivates the goals of this thesis and discusses related work. Next the ESB framework is introduced at the theoretical level, before moving on to details of an implementation in the subsequent chapter. After this, the evaluation chapter presents two case studies and several algorithms that can be used with ESB to bound reasoning. These are discussed with respect to the hypothesis and goals of this work. Finally the main contributions here are detailed and possible areas for future work enumerated.

Chapter 2

Background

The area of multi-agent systems research lies at the intersection of artificial intelligence and distributed systems. Agents can be considered reactive and deliberative distributed systems, as they continually interact with their environments to achieve some task in a manner where the computational reasoning is distributed. The aspect of MAS that makes agents unique and distinct from traditional distributed AI approaches is that although the agents and indeed the system are dedicated toward some goals, they have autonomy in the methods by which they are achieved. In the view taken in this thesis, the agents have percepts and internal beliefs that they can manipulate themselves to support reasoning on actions towards their goals.

This thesis describes a generic framework for social reasoning, so it is first necessary to explore what social reasoning is and the current related MAS literature. This review has not only motivated but also guided the development of the ESB social reasoning framework. Furthermore, study of this area of MAS provides the reader with an appropriate background knowledge in the area to appreciate the contribution here and its place in the larger volume of work.

The first and largest section of this chapter will present the social reasoning approaches used in MAS. As the goal is to capture social reasoning in a general way, each different area of social reasoning is presented in turn, with discussion of the key properties. This is the body of work related to the goals of this thesis. The next section will summarise the main themes present in agent social reasoning and how they have motivated and guided the work presented in this thesis. After this comes work related to the approaches taken in the thesis. This is literature related to the proposed solution

more directly, rather than the obliquely related work of the earlier section. The chapter concludes with a summary of the background presented here, setting the scene for the main body of the thesis.

2.1 Agent Reasoning Methods

The aim of this review section is to cover the body of work related to the goals of the thesis for two reasons. Firstly, it sets the scene and provides motivation for the hypothesis and secondly, existing social reasoning methods inform the decisions made to create the general purpose ESB social reasoner presented here. Accordingly this is a broad review, covering the whole spectrum of agent reasoning as far as it is directed toward other agents. As well as this, implemented agent reasoning systems are also of interest as creating an implemented social reasoning mechanism is a key concern of the hypothesis for both practical and evaluation purposes.

Broadly speaking, the areas covered are an attempt to characterise social reasoning in general terms. The eventual goal is the loose classification and evaluation of agent reasoning presented in the summary section 2.1.3. The wide variety of techniques are grouped into the following categories:

Practical reasoning – For an agent to deliberate and choose appropriate actions to achieve a social goal, it will still require the ability to reason practically. In addition the field of practical reasoning is (relatively) long-established and so may provide cues for the development of social reasoning.

Cooperative Agent Techniques – Cooperation between agents is the most important reason why they should interact at all, so it is important to consider the properties that might be required for a general social reasoning system.

Communication based reasoning – Integrating the effects of communication into agent reasoning is a hard problem. Results of interactions, unlike manipulating the environment, are rarely immediately obvious. As reasoning about communication is key for a social agent, it makes sense to investigate existing approaches.

Agent Societies – Many have thought to lift much of the burden of social reasoning by enforcing rules or norms on agents acting together. As this is a possible means of social control of agents it has a significant impact on the agent reasoning

process, even though most approaches consider the agents at the system rather than individual level.

Opponent modelling and trust – Approaches used by agents to evaluate and account for others' behaviour form a necessary part of any theory, therefore a brief study is included here.

The topics covered are chosen as they are the main approaches for social reasoning described in the literature. For the purposes of background, the specifics of each technique are not important, only their characteristics. This work does not directly follow on from any existing work, but the background of the approaches taken to construct the whole will be presented later, in the appropriate place.

It is important to distinguish the theoretical approaches, that describe what an agent should deliberate about, from the approaches concerned with implementation and how this deliberation is carried out. The distinction is that theories deal with *what* to deliberate about and the computational approaches deal with the *how*. Both will be surveyed, the ESB framework presented in this thesis has overlaps with both approaches, as it is an attempt to bridge the gap.

2.1.1 Practical Reasoning

Practical reasoning approaches are worthy of investigation as any agent reasoning socially will also require to reason practically about what actions to carry out. Some well known work is inappropriate to consider in detail here. Reactive agent control utilises sets of simple percept-action pairing rules which combine to give an agent a complex emergent behaviour from simple easily modelled rules. Deductive reasoning approaches, or "agents as theorem provers" (Wooldridge [2002], p49) where the agents use a set of logic rules and conduct inference to deduce actions are more of a class of implementations than a reasoning approach in their own right. For example, a BDI agent could also be implemented as a deductive agent. Soar [Soar Group, 2009] is a framework for cognitive reasoning that specifies how an agent's reasoning processes work, but not about what they reason over. In addition to these brief samples, there are many hybrid architectures for reasoning, which tend to combine reactive and more deliberative planning based approaches in some way.

2.1.1.1 Belief Desire Intention (BDI) Models

The BDI model of practical reasoning [Bratman et al., 1988] was proposed as a theoretical architecture for limiting an agent's reasoning based on its intentions, in a bid to combat the problem of an agent's bounded resources. There are many good detailed descriptions of BDI; as it is one of the main approaches for practical reasoning in the MAS field, it appears in many standard MAS textbooks [Wooldridge, 2002, Weiss, 1999, Nagi, 2001, Bratman et al., 1988].

The intuition behind BDI theory is that reasoning can be constrained to that consistent with an agent's goals. Practically, an agent may wish to drop or revise intentions, but this notion can still be used to bound reasoning. The operation of a BDI agent can be simply described at a high level. An agent has a set of current beliefs, some desires and a method of means-ends reasoning to generate consistent methods to achieve intentions. A belief revision function updates the beliefs, which along with the agent's current intentions feed into an options function. This gives an agent its current desires, which along with the beliefs form the input to an action selection function. There are a few notable benefits of this algorithm. Desires can be revised based on new information – this is what gives the agent its flexibility and autonomy. An illustration of this advantage is that the agent can be opportunistic. For example a delivery agent notices the addressee of a letter it planned to deliver later walking past and it can adopt an intention to give that person the letter now, saving effort later.

The implementation of BDI is a separate issue to the theory and logics, with most implementations following the style of PRS (Procedural Reasoning System, Georgeff and Lansky [1987]). The key point about the operation of PRS and other BDI implementations is the handling of the plans. Whilst BDI theories and logics presume a top down approach, with a means-ends reasoner and action selection function to create action, PRS takes a more pragmatic bottom up approach. It is assumed that the designer will know the actions the agent is capable of and these can be combined in several pre-decided plans to achieve certain goals. The reasoning then reduces to a problem of plan selection, where the agent operates by choosing plans consistent with its beliefs and intentions (initial goals and any subsequent sub-goals).

To demonstrate the working of a BDI in more detail, Figure 2.1 shows an overview of the Agentspeak(L) interpreter as used in the Jason [Bordini and Hübner, 2009] implementation. Jason is used as part of the implementation in the evaluation part of the

work in this thesis, so describing a BDI reasoner in more detail here has utility beyond this section. The operation of this interpreter is roughly as follows. The inputs are perceptions, which are used in combination with beliefs by the BRF (Belief Revision Function) to update the agent's belief base. The core mechanism to motivate action is the notion of "event" which could be a belief revision, or internally generated goal. Once an event is chosen to act on, it is unified against the plan library, to get the plans relevant to the event. The applicable plans are then selected based on their context (which plans are applicable given the agent's current beliefs). From this set, a plan can be chosen to be intended. Action selection itself depends on choosing one (of possibly many) current intentions to execute – and so the cycle repeats. The significant abilities of a BDI agent are captured in this cycle. Plans (adopted as intentions) may generate new events leading to new plans being adopted, this is the mechanism used for plan refinement or reconsideration. The intentions are managed as a stack, so an agent can be engaged in actions toward more than one goal simultaneously.

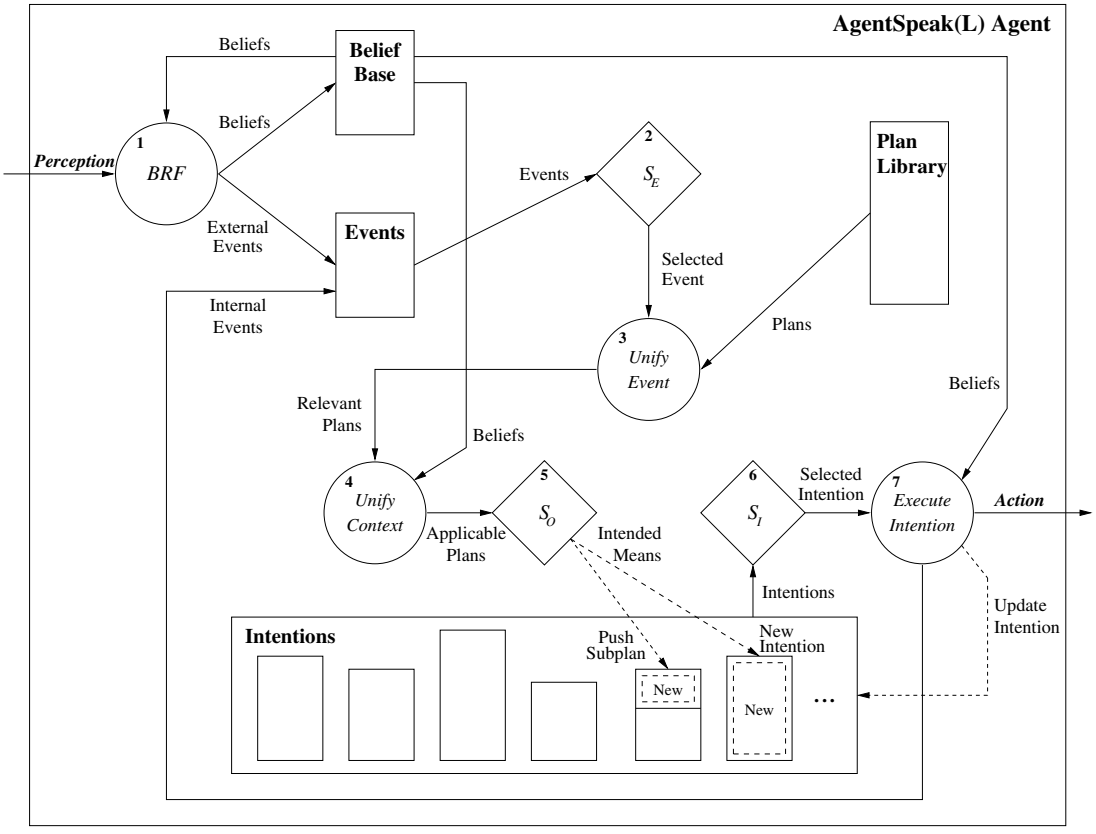

Figure 2.1: AgentSpeak(L) Interpreter Cycle (diagram from Machado and Bordini [2002])

As we are looking toward the BDI model for inspiration, a natural question is what

makes it so successful? The panelists in Georgeff et al. [1999] suggest a few reasons. It is based on a model of human practical reasoning, has been implemented in several architectures and successful applications and has logical semantics that have been widely studied. The fact that it has been implemented and applied to applications could be seen as an indication of its success, rather than a reason for its success however. The implementations also show that BDI is a workable model, below are several properties that make it suited to implementation and further study:

Simple Idea. The basic idea behind BDI is very simple and intuitive, as it is based on a simple model of human reasoning [Weiss, 1999].

Aids the design process. One of the reasons for the usefulness of the BDI approach in implementations is that it guides the agent designer as to how to structure the agent. As Wooldridge notes "it gives us a clear functional decomposition, which indicates what sorts of subsystems might be required to build an agent" (Weiss [1999, p60]). Most BDI implementations have a fairly similar basic reasoning loop.

Theory does not constrain implementation design. Although the general BDI theory guides the implementation design it does not overly constrain the structure. For example Cavedon and Tidhar [1996] give an example where a team of dinner preparing agents only have beliefs and intentions. This is because the agents have a fixed goal (to prepare dinner) so they have no desires based on their beliefs and they are dedicated toward achieving their intention. Flexibility in how the BDI theory can be adopted for implementation as in examples such as this are obviously an important reason for its success in implemented systems.

Stepwise refinement of plans. As the reasoning follows a looping refinement process ,it means that an agent needs only a partial plan to start acting, further information and refinement necessary to complete its goal can be found during execution. The refinement mechanism of BDI that allows this also avoids problems with overcommitment to plans doomed to failure and helps build an agent that can react in real-time. An example of this is the agent designed for space shuttle maintenance and fault diagnosis presented as an example of a PRS agent by [Georgeff and Lansky, 1987].

Easily formalised in logic. There have been many BDI logics created (Rao [1996] gives several examples). They can be used to model systems for specification

and design purposes, to validate claims about agent systems [Wooldridge et al., 2006, Rao, 1996] and can help provide theoretical solutions to practical problems. The logical representations are one of the most important properties of BDI but on their own they are not so useful. In defining their logic of norms and obligations for BDI, Dignum et al. [2002b] argue that logics cannot give any insight into implementations without a clear link between theory and practice. It is the existence of logics alongside implementations that help make BDI successful.

The points made above about BDI are certainly worth bearing in mind for any social reasoning theory or framework. The same headings could be applied to nearly any successful theory in computer science; all that would change are the examples and justifications.

2.1.2 Social Reasoning

Social reasoning, like many terms in the field of MAS, is overused and subject to different definitions. For the purposes of this thesis, it is concerned with how an agent reasons about the outcome of its interactions with others and takes the actions of others into account for its own reasoning. Again as with practical reasoning there is a distinction to be made between the theoretical *what* to deliberate about and the implementation concerns of *how* to carry out this deliberation. In this case, implementation concerns are of interest, because they may highlight particular social reasoning problems. There is also the notion of specification to consider namely some schemes aim to define the behaviour of an agent, or system of agents, without providing a route to the implementation of this behaviour.

Here the focus is on how each approach affects social reasoning. It may be that the theory has a direct effect, describing the deliberation process that an agent should use, or it may be a more subtle influence. The effect may be implicit in the design. For example, the richness of an agent's communication language and percepts will affect the richness of its reasoning process, but this may be accounted for in the agent's design. For cases such as these the question is would a more complex agent explicitly reasoning about the effects of communication be more useful? Would it even be possible to design such an agent? It may be that even if design or implementation of such an agent is not feasible, a specification that could capture all the factors in the design, hidden or

otherwise, could prove useful.

In a lot of cases, schemes presented for reasoning in fact do the opposite; they are schemes to remove the need for agents to carry out individual reasoning by designing them in such a way that the interaction problem is solved at design time, rather than run time. This is obviously a less feasible approach in open systems, but can still be done to some extent or for certain classes of interactions.

Broadly speaking there are only a few basic classes of joint action an agent may engage in and so needs to reason about. Van Dyke Parunak et al. [2003] attempt to formally define these, mostly in terms of observations about a group of agent's actions and the information shared between them. The categories they define are as follows:

Correlation is the most general term for agent interaction and is defined as non-zero joint information between the agents.

Coordination is defined as correlation with communication causing it. The types of coordination are defined on the style of communication between the agents.

Cooperation and Contention are classed as correlation where the intent of the agents is to either help or hinder each other. These intentions are not related to the communication methods, hence it is not a type of coordination by their definition.

Collaboration is where agents communicate with a mutual intent to cooperate, resulting in coalitions.

Congruence is the term they use when agent goals match system goals and coherence is the relation between agents that results in this.

The key point to note here is that there are these different forms and some schemes model only some of these interactions even though agents may be capable of several. For example although agents may use the principles of Joint Intentions (introduced in Cohen and Levesque [1991]) to co-ordinate as a team, they may also interact competitively with other agents and this may be governed by a different theory.

2.1.2.1 Cooperative Agent Techniques

Cooperation between agents is very important in MAS. The goal of all MAS is that groups of agents may work with or against each other, but in general using the services of others to achieve greater utility than they could alone. This section is about systems

for agents to achieve their goals, or team goals. To discuss some of the concepts two well known approaches are referenced – Joint Intentions [Cohen and Levesque, 1991] and Generalised-Partial-Global-Planning (GPGP, Decker and Lesser [1995]).

There is a difference between how agents agree to cooperate and the actual process of achieving something through collaboration. There is also a distinction between working for team benefit or as a team for individual benefit. For example, a team of agents may work together to play robotic soccer. Here the goal is literally a team goal, it is the team that benefits. An example of an agent cooperating with others for individual gain could be an agent forming a coalition with others to negotiate a bulk discount on a product or service. Through cooperation with others it achieves a utility it could not achieve alone. This section considers only those methods for how an agent system should be designed to allow agent reasoning for cooperation, not the various types of systems for encouraging cooperation described below (norms, virtual organisations etc.). Even more specifically, systems are considered that are designed for explicit cooperation.

Joint intentions can be most basically described as the necessary goals and intentions for agents to work as a team. They define the mental states agents must go through to jointly intend a goal. This is built on the notions of persistent goals and persistent weak achievement goals (PWAG). When an agent has a persistent goal to achieve A, it will intend A and try and achieve it until A holds, is impossible or no longer a relevant goal. A PWAG is held with respect to a goal and another agent Y. It is the same as a persistent goal, except that once it is complete or abandoned, the agent also holds a persistent goal to notify Y that it has succeeded, become irrelevant or impossible. Agents have a joint commitment to an action when they hold interlocking PWAGs with each other to achieve it and it is a joint intention when they both intended this state of affairs. A more formal discussion of the necessary speech acts and mental states to achieve joint commitment is presented by Kumar et al. [2002]. The below definition is for a joint persistent goal as described in [Kumar et al., 2002] and gives an illustration of the type of concepts involved in joint intentions:

$$\begin{aligned}(JPG\ x\ y\ p\ q) \quad \triangleq \quad & (MB\ x\ y\ \neg p) \wedge (MG\ x\ y\ p) \wedge \\ & (UNTIL[(MB\ x\ y\ p) \vee (MB\ x\ y\ \Box\neg p) \vee (MB\ x\ y\ \neg q)](WMG\ x\ y\ p\ q))\end{aligned} \tag{2.1}$$

This defines when two agents hold a joint persistent goal (JPG) toward some action p whilst some relevancy condition q holds. It says that they must mutually believe

(MB) $\neg p$ and have a mutual goal (MG) to bring it about. Also p will be a weak mutual goal (WMG) until it is mutually believed to be achieved, never possible or becomes irrelevant. As well as illustrating the sort of formal definitions typical for this type of work, this highlights another property. The mechanism is reliant on a notion of mutual belief, which can never be achieved in practice without perfect communication. The implication of this is that agents must either be designed to work together so their mental states can be known (or estimated) or there will be a level of mental modelling or assumptions about the other agent's reasoning.

A different, lower level, approach to achieving cooperation between agents is GPGP [Decker and Lesser, 1995], which focuses on coordination between agents – necessarily sharing a common design. The approach here is a set of scheduling coordination mechanisms, of which some or all can be used as is appropriate to the task. Their purpose is to allow for the local task schedulers in each agent to coordinate to ensure tasks are not done twice or agents do not interfere with each other. The mechanisms presented to ensure that all agents scheduling constraints are met include updating non-local viewpoints, communicating the results of actions according to various properties and ensuring coordination constraints between multiple agent's actions are upheld. This is obviously a much lower level of abstraction to Joint Intentions (JI) and thus seems less like agent reasoning. It is instead a set of guidelines for a designer of co-ordinated agents, taken into account during design and implementation. Compared to this JI is a theory, that although also applied at design time, is more generally applicable. The principles could be used by agents in an open or closed MAS and the agents need not necessarily be homogenous. To apply JI needs further "reasoning", hence the development of the theory into the communication protocols presented by Kumar et al. [2002] to apply it.

This is only one example of differing levels of abstraction for cooperative agent techniques. Joint Intentions specifies what teamwork is, the necessary properties of agents and their communication to achieve this, whereas approaches such as GPGP aim to provide a method for many agents to achieve a joint task and specify in more detail how an agent should reason. This does not mean comparisons cannot be drawn. It is possible for a GPGP based architecture to also meet the specifications of Joint Intentions. A problem with these approaches is that focus on mutual belief states can present problems, as often communication will be unreliable and so the beliefs of other agents may be uncertain.

The key similarity between both these approaches and why they are useful is that they give guidance as to how an agent should function to work cooperatively. This can aid the designer of agent systems. The emphasis is on systems, as for the principles of Joint Intentions and GPGP to hold and agents in the system to collaborate, all agents' reasoning systems must have certain common properties so that they can predict the mental states, and so actions, of others. The constraints with GPGP are even greater – it is assumed that each agent has its own local planner and will integrate actions shared from other agents as commitments in the same way.

Due to these constraints on design, GPGP removes the need for agents to reason about teamwork. By designing the agents and their planners such that plan actions are shared according to GPGP theory, building up global plans, it removes the need for agents to reason about how other's actions contribute to the team goal. This is accounted for in the planning algorithms. Indeed it could be argued that the theory of GPGP is one of distributed problem solving that can be applied to agent systems, rather than a social reasoning theory, as often it is applied to resource or action co-ordination problems.

The main use of Joint Intentions is to guide the design of communication strategies for team systems. For example the STEAM architecture described by Tambe [1997] and the work of Kumar et al. [2002] use the principles of Joint Intentions in the design of the communication protocols. Smith et al. [1998] describe the use of JI to design the semantics of communication acts. Just because the current applications of JI theory are at the system design level, does not mean there is no value in representing the principles at an agent reasoning level. The reason JI has been applied to closed systems is that the communications protocols used need necessarily be used by all agents in the team for the actions designed for the agents to perform as expected. However, one could envisage a more complex agent in an open system, with knowledge of JI theory, negotiating with potential team mates to agree on a JI based protocol for co-ordinating team activities. This could allow forming teams in open environments with previously unknown agents. An example of using ESB to represent JI in a general way for agents to achieve specific team based goals is presented later, in Chapter 5.

A flaw in the theory of Joint Intentions is noted by Jennings [1993]; the theory defines all the actions and goals in terms of individual agents and their commitments. There is no explicit notion of groups of agents, or group goals. This could hinder design of such systems and means that JI cannot be used to reason about groups as a whole.

2.1.2.2 Communication

Social reasoning often involves reasoning about communication with others, as it represents the most explicit form of information transfer possible. This is in contrast to an agent's actions, which may implicitly transfer information to others. The major issues for communication in MAS are what to communicate, to whom and how often, dictated by the protocols for communication. At a lower implementation level, the specific languages, protocol implementations and ontologies used are of interest too. As the topic of interest here is social reasoning approaches, the more practical concerns of ontologies are of less interest.

Whilst the survey presented by Kone et al. [2000] can no longer be considered "state-of-the-art" due to its age, the authors still make interesting points regarding the properties that agent communication languages may have. They highlight the importance of assigning semantics to speech acts. For an agent to reason based on communication and inferred mental states of other agents, it is obviously an easier and more reliable task when the semantics of speech acts are known. In this case, the communication imparts some knowledge of the other agent's beliefs. However they note that general semantics for communication might not be possible, dependant on domains and the effect of context on communication meaning.

The rest of this section considers how communication can affect social reasoning in more specific terms.

2.1.2.2.1 Social Reasoning Based on Communication

Social reasoning could be carried out based on the communication language available to the agent for a particular class of interactions. Features of the language used, or communication architecture may affect an agent's interactions. For example, in a scenario where no private communication is possible, it may be harder for the agent to form teams to act against others. An example of features of a language being used to guide reasoning about interactions is the work of Osman and Robertson [2007] which shows how an agent can prove properties of protocols for interaction at run-time. The approach presented uses a model checker to verify if interaction protocols are compatible with deontic trust specifications. If an agent was to base its reasoning on whether or not to interact based on these proofs and reasoning about what to prove, then this

would represent social reasoning based on communication. However this would be a step further than what is presented, as this work assumes that there is a given strategy for trust. An example given is auction protocols that encourage truthful bidding. These strategies could be seen as the core of the social reasoning on communications and the model checker is one approach to make these useful. Ideally an agent would reason about these strategies itself and so be able to select different ones, or even evolve them to meet its current goals or environment.

In his thesis, McGinnis [2006] presents a formal theory and implementation for agents able to modify their own protocols for interaction, through the use of distributed protocols that can be communicated to and understood by other agents. This is another piece in the puzzle of social reasoning about communication, as while the above work presents a method for an agent to reason about protocols, this is a method for agents to negotiate, change and communicate their protocols. The issue of how to reason still remains. Although examples of how the method works are presented, the knowledge and reasoning as to how and when to change protocols are still specified by a designer. Agents only make changes to a protocol when the current dialogue state matches an internal knowledge base of conditions to change the protocol and the protocols are initially synthesised by the agents through existing domain rules.

2.1.2.2.2 Speech Acts

Speech act theory [Searle, 1969] is based on the premise of treating communication as action, that an agent can change the world not only with its physical actions but also with utterances. There are a wealth of related approaches, due to the popularity of the notion that practical reasoning and planning techniques can be re-used for social interaction. This section covers a selection of these and their affect on agent design. A model, based on actions that only succeed in the presence of certain preconditions, is an obvious candidate for logical representation and reasoning and not a new idea, stemming from philosophies of human practical reasoning. Cohen and Perrault [2003] (originally published 1979) introduce the idea of using speech acts as operators for classical planning and discuss the inadequacies and properties of such operators with respect to their suitability for composition into plans representing interaction protocols.

A major organisational paradigm where agents' utterances can have effects is that of

agent organisations and electronic institutions (see section 2.1.2.3). Here an agent with appropriate authority may be able to change the structure of the organisation, or even its rules. For example, Demolombe and Louis [2006] present an extension of the FIPA ACL formalism to institutional speech acts, where agents can create obligations and assign roles in an institution. This extension to FIPA ACL provides a theory of how utterances could effect institutions, but it would only serve as a guide to create a language and from there an agent implementation. Another similar approach is discussed for norm-based virtual organisations by Boella et al. [2005], with a conceptual model where agents may use speech acts to give other agents permissions and create obligations. Again this is only a model, not even fully formalised, presented as preliminary research to aid agent design.

Rovatsos [2007] presents the outline of a formal theory to a slightly different approach for dynamic agent systems involving speech acts. Here it is the agent that changes, rather than the society, through the use of dynamic semantics for speech acts with a system where change is based on the expected and norm-compliant behaviour of agents. The need for this is to account for agents with changing behaviour (for various reasons). This paper stands out, in that it considers the dynamics of agent reasoning as opposed to static systems.

The above examples represent commitment based communication, where utterances are defined in terms of normative rules and contracts between agents. Nickles et al. [2004b] note that models of communication tend to be either commitment based or mentalistic. In mentalistic approaches, utterances are in terms of agents' mental states, defined by the updates to beliefs and intentions (and therefore implicitly actions). The problem with these approaches are that they require some model of the reasoning process of other agents. However it may not be possible to extrapolate behaviours from beliefs in an open MAS scenario. Furthermore, Nickles et al. [2004b] claim that commitment based approaches are an oversimplification, as they deal with small sets of normative rules and so cannot handle indefinite communications or malevolent agents. The proposed solution is to define communication semantics in terms of expected future agent actions, these can be learnt based on observed communications and the resultant actions.

One example of a mentalistic approach is in the situation calculus based approach of Khan and Lesperance [2005]. They describe how requests affect an agent's intentions

and for example, the preconditions for an agent to *inform* another of ϕ:

$$Poss(inform(inf, agt, \phi), s) \equiv Know(inf, \phi, s) \wedge \neg Know(inf, Know(agt, \phi, now), s). \tag{2.2}$$

The description of this given by the authors is as follows:

> "the agent *inf* can inform *agt* that ϕ, iff *inf* knows that ϕ currently holds, and does not believe that *agt* currently knows that."

Similar formalisms are given for how other communications affect an agent, in terms of pre- and post-conditions. The communication language available defines how an agent plans; the language extends the actions available though the choice of interactions it presents. The social reasoning that an agent can perform is limited by the expressiveness of the language that it can communicate with. In this work only methods to allow simple agent cooperation are presented, with the assumption that one agent plans for all others.

2.1.2.2.3 Argumentation

For the topic of social reasoning a complex and interesting form of interaction is argumentation. It allows a rich reasoning process, as the reasoning of others needs to be considered to counter their arguments. This is in contrast to alternatives such as game theoretic and auction based approaches to negotiation which can be tackled through reasoning about the interaction mechanism.

Argumentation is carried out by exchanging beliefs and logical inferences to counter beliefs and lack of knowledge in others. This can allow an agent to influence another's internal state – indeed this is the very purpose of argumentation. At some level this shares similar problems to opponent modelling techniques (see section 2.1.2.4), as it requires a model of how an opponent constructs arguments to allow for countering them. Argumentation techniques are a rich field of AI in their own right, their applications to agent systems is mostly in terms of protocols and their like for specifying types of argumentation, rather than work about how agents should use arguments in their reasoning.

Recent work has started to consider how an agent can use argumentation for action directed reasoning in specific cases. For example Belesiotis et al. [2010] consider how pairs of agents can use argumentation to reach agreement on a shared plan. In

this case, although there is a test implementation, the focus is on creating a protocol with the desired properties. The result is closer to classical planning than a general reasoning system, but with additional dialogue between the agents.

Rahwan et al. [2003] note that argumentation based negotiation (ABN) requires a richer communication and domain language, to enable agents to exchange points of view with the necessary information on the reasoning process. This point is in support of the conclusion made in the previous section that the sophistication and scope of agent social reasoning is limited by the communication available. The issue of trust is brought up by the authors as it can have a direct bearing on any negotiation. This is only one obvious example of how for any complete system an agent must consider not just one social reasoning problem, but several. This is one of the motivations for the proposed separation of social reasoning and the creation of a general social reasoning to allow combining approaches more easily.

2.1.2.3 Agent Behaviour Constraints

This section is about approaches that aim to simplify or in part solve some of the problems of open MAS by behaviour constraint, either through the use of hard-and-fast rules, possibly with sanctions for disobedience, and more flexible schemes such as social norms. This moves the burden of some of the social reasoning problem to the system, rather than the individual agent. This can simplify agent design and often moves some of the burden of reasoning onto the system designer.

2.1.2.3.1 Electronic Institutions

Esteva et al. [2000] describe the concept of Electronic Institutions (EI). The purpose of EIs is to handle the complexity of allowing interactions between heterogeneous agents in an open MAS setting by constraining them to a set of rules. Roles with their fixed set of actions allow an agent to reason about interactions, by restricting the scope of an unknown agent's behaviour. A dialogical framework defines a set of allowable illocutions in terms of a common ontology, communication language and knowledge representation language. This is the key feature which allows for the interaction of *heterogeneous* agents. The formalisation of these roles and language must be done by the agent designer and all agents in the system must be aware of them for there to be

benefit. This implies that an agent's design will be influenced to a greater or lesser degree by the EIs it is intended to operate in.

The advantage of EIs is fairly obvious: they provide a way for agents in an open environment to interact, mostly by providing a common interface between agents. Implicit in this common setting is the need for a trusted mediating agent to see that norms and rules are adhered to. For example in the auction case the system breaks down if the winning bidder can simply decide not to pay. Another disadvantage is that the rules of an EI are generally static, as they are designed by someone. However this need not always be the case, Boella et al. [2005] and Demolombe and Louis [2006] both describe systems of speech acts which could be used to allow agents to change institutional rules, as mentioned in the previous section on communication based reasoning.

2.1.2.3.2 Virtual Organisations

The term "virtual organisation" (VO) covers a wide variety of agent based systems, including the team based systems described previously in section 2.1.2.1. As they take their inspiration from human organisations, VOs tend to focus more on the control relationships between agents and how these define the behaviour of the group.

Commitments between agents are a common realisation of the organisational relation between different roles. Both Carabelea and Boissier [2006] and Jennings [1993] consider how commitments can be used to define the behaviour of a role in a VO (although Jennings [1993] lists VOs as an example of *implicit* commitments, the work in Carabelea and Boissier [2006] makes them explicit). Grossi et al. [2005] argues that a more formal model of these relationships in VOs is required and presents a logic based formalism based on power (as in authority), co-ordination (in terms of knowledge and information) and control (in terms of monitoring and recovery of actions). The purpose of this logic is not for an agent's social reasoning capability to be improved, but rather so that organisational structures can be reasoned about in terms of their effect on agent reasoning. However no example of applying these techniques to an implementation is given. Despite this, the concept of commitment between agents is key for them to cooperate in an open environment. The concepts presented are worthy of study – any agent social reasoning will be required to represent them in some way.

2.1.2.3.3 Social Norms

Norms are rules governing the allowed actions and states of an agent society. In general they are expressed as obligations, prohibitions and permissions. These can allow for easier agent reasoning, especially where a system includes heterogeneous agents with differences in goals, autonomy and reasoning power. Additionally they can be used to control the actions of an agent society as a whole and direct it toward a specific task.

Work on how norms affect an agent's social reasoning is mostly in terms of the influence of normative systems on regular models of agent reasoning. Dignum et al. [2002b] aim to add social concepts, such as norms, into the BDI model of agent reasoning, to create a theoretical general agent reasoning model, rather than a discrete social or practical reasoner. This is based on the work presented in [Dignum et al., 2000] and [Conte et al., 1999] which discusses an extension of BDI logics to include the concepts of norms and obligations. They present only a concept of how this might be done and show through a theoretical formalisation that it is a promising approach.

The main observation of their work is that most agent architectures treat desires as goals that cannot be mutually exclusive. However, they argue that there is a need for distinction as desires may conflict, as they may not be rational. This could happen in the case of norms and obligations, as they effectively represent external desires the agent wishes to satisfy. These could be inconsistent with the agent's desires, for example in the case where breaking a norm is desirable but incurs an undesirable sanction. Their proposed solution is to reason over desires according to a preference ordering based on their source.

There are three utilities to consider in the agent's reasoning; personal utility (related to desires), utility for the society as a whole (from norms, as they are assumed to benefit society) and what the authors refer to as "the degree of social cohesion" – failure to meet obligations will fragment society.

Although the model in [Dignum et al., 2002b] suggests how an agent may include norms and obligations in its reasoning, it does not suggest how an agent may reason about norms and obligations themselves, which is surely necessary for social reasoning. So as a result it simply represents how a practical reasoning agent may take into account social factors, rather than a socially reasoning agent.

This concept of a *norm-autonomous* agent is described in Castelfranchi et al. [2000].

This is an agent that can reason not only using norms to direct goals, but also about the norms themselves. This is important, as it allows an agent to decide when to violate or adhere to a norm and consider the consequences of adopting a proposed norm. There exist implemented norm-autonomous systems [Kollingbaum, 2005], but although a method for reasoning about violating norms is presented, the exact methods used to determine use (or not) of a norm are left to the agent designer.

Another logical theory of decision making in normative systems is presented by Boella and van der Torre [2005], specifically targeted at the problem of trust. It follows a similar pattern with a system's norms and obligations influencing an agents goal selection, but also considers the issues of trust and the possibility that an agent may wish to violate norms. The key difference is that the norm system itself is represented as an agent and the effect of the norms is modelled by a game between this system agent and the agent doing the reasoning. This is the key difference that allows one of the agents described here to consider violating a norm. It has no concept of a norm as such, merely an obligation to the agent (the norm) and an action that it may perform if it is violated (the sanction).

One issue surrounding norms is change. Norms describe usual behaviour, so if more and more agents ignore a norm, by definition one would expect it to cease to be a norm. In an open system it may be desirable for agents to be able to create and modify their own norms, rather than the system designer. Ultimately in open systems it is desirable for agents to have the ability to adapt the system they are in, both actively through modifying the rules if they have suitable authority and passively, as the majority follow or ignore norms.

In many situations it is desirable for norms to be adaptive and change with time. They may change due to direct influences from agents attempting to change the structure of the society they are in, or it may be an indirect change with norms arising or disappearing depending on an agent's behaviour patterns. There is little work into how this affects an agent's reasoning however, in terms of how they may reason for change or indeed a dynamic norm system could be implemented. Boella and van der Torre [2007] present one approach to this problem, with their model of norm change. They present a "social delegation cycle" where through communication agents' desires give rise to agreed social goals and negotiated norms to achieve these goals. In this way the agent society can generate new norms. This only considers a subset of the possible norms an agent system might hold (all the norms are as a result of agents' goals for

example) but it is still a useful illustration of how norms may change dynamically.

2.1.2.4 Opponent Modelling

Reasoning could rely on an agent using a model of its opponents. For example, Saha et al. [2005] describe a buyer using details of past transactions to create a model of a seller's strategy and so attempt to offer the minimum acceptable bid. Opponent modelling techniques may not necessarily be solely applied to competitors. The GPGP scheme (section 2.1.2.1) requires all agents in the team to reason the same way. In some sense they are modelling other's actions, though not explicitly in this case, the modelling happens at design-time.

The rationale behind opponent modelling is that the model need not function internally the same as the target being modelled, so long as it predicts a close-enough output for a given interaction or input. Carmel and Markovitch [1996] describe a game theoretic approach based on this, that of learning finite-automata representing the target agent. Their intuition is that useful properties of another agent's reasoning can be captured as a deterministic finite-state automata (DFA) modelled on observed past behaviour. This model can then be used to predict future behaviour. This has several appealing properties. It is a computationally simple model, but also has several obvious drawbacks. It is based on game theoretic approaches, assuming an agent playing to maximise its payoff. A devious agent may lie, or deliberately mislead, or simply not be "clever" enough to make what appear to be the best choices.

There are different levels of abstraction that can be considered as opponent modelling. Carmel and Markovitch [1996] in their approach choose to formulate the problem as a repeated game, where the reasoning is represented by the learning algorithm to generate the DFA. The buyer modelling a seller's decision function presented by Saha et al. [2005] also reduces to a type of learning problem. Here the goal is to model a decision function of a seller agent, which is reduced to the problem of estimating a probability function (that the seller will accept a bid) from a series of yes/no results about function data points (the previous transactions). The low level approach has the benefit of being suitable for implementation. The algorithm presented in this example has favourable computational cost and they present experiments to show that it works in practice, and can back up claims on convergence to the desired functions with proofs.

However, these approaches only simplify the decision process and any agent would

require the reasoning integrated into a more general reasoner to be useful, to guide action selection etc. These processes only help for very specific defined cases and can be considered more in terms of general AI techniques than MAS. Much agent reasoning is also considered at the higher symbolic level. This is required to reason more generaly or about more abstract concepts that cannot easily be reduced to a simple numerical problem.

Vidal and Durfee [1998] discuss the issues around learning nested models of other agents and when it's worth reasoning with or without this information. This follows from work in Vidal and Durfee [1996] where they explore the utility of the level of nesting in modelling another agent's reasoning. In this example the problem tackled considers not just another agent's reasoning but also its model of the first agent (and so on recursively) – the nesting. The scenario used is that of a digital economy, with markets and auctions. The balance they explore is that of an agent's trust in the market mechanism versus its own strategic reasoning about other agents' strategy. As far as the goals of a general social reasoner are concerned, the interesting result here is that there are indeed cases where this recursive modelling is useful – and some form of meta-reasoning over when to apply this is also required.

2.1.2.4.1 Trust

Trust matters, as unless the agent has perfect information regarding the mental processes and inputs to another agent, there will be some measure of uncertainty as to its actions – therefore to some degree it must trust the other agent.

Ramchurn et al. [2004] survey the issue of trust and identify three main interaction problems which are governed by some notion of trust:

1. Protocols for interaction (protocols may remove the need for trust, or account for differing values of trust).
2. An agent's reasoning on who to interact with (who is trusted?).
3. Reasoning about when to interact.

Of these it is the who and when which are most clearly part of the problem of social reasoning. Interaction protocols are a more concrete approach compared to the other two, a technique to implement the designers reasoning. This relates to another general

theme they identify, that of system level versus individual trust. Trust at the system level is means to enforce trustworthiness designed into the system, whereas individual trust is a way to interact with other agents without complete information about their intentions.

Although creating and maintaining some measure of trust and ensuring the mechanism is reliable against subversive opponents is a tricky problem, without some way for an agent to link trust into its reasoning it is worthless. Patel et al. [2005] describe how the CONOISE-G project applies notions of trust to the problem of forming VOs to provide web services in an open environment. This problem is tackled with a combination of trust at the agent level and system level. At the system level there is a policing system to control breaches of contract – giving the agents trust in the system. There is also a reputation system, which bridges the gap between the system and the agent to some extent. At the individual level each agent generates a trust metric for each interaction and associates with it a confidence, if it is not confident it can solicit an external opinion from another agent or a reputation broker. These reputation brokers are agents with the sole task of collating individual views of agent's reputations and answering queries based on this information.

Despite the varying levels of abstraction in trust and opponent modelling, it can be considered as part of a larger more general problem. A framework to separate out social reasoning should consider the more general problem of how to integrate external influences on the decision process. Trust metrics or models of other agents' behaviour are merely specific instances of this more general requirement.

2.1.3 Summary

Thus far this background chapter has covered the wide variety of agent reasoning techniques that can be considered as social reasoning, introducing each and the key points surrounding it as far as the reasoning is concerned. This can also be considered the related work, in that it is work related to the goals of this thesis as presented in Chapter 1. Before continuing with presenting the work related directly to the implementation and theory presented in this thesis in section 2.2, this section first summarises the review so far. The state of social reasoning as presented here provides the main inspiration and motivation for the thesis and so this section draws conclusions and motivates the work.

Table 2.1 provides a summary of the categories of reasoning techniques covered and a rough classification. Each column lists a criteria along which the approaches have been assessed, with a scale showing the extremes:

Level of Abstraction refers to how theoretical the approach taken is. The scale could be thought of traversing a spectrum from theory, through empirical approaches, to implemented applications. [1]

Generality comments on how general-purpose the approaches are. Can they be applied to any domain, or are they restricted to specific systems?

Focus considers when the reasoning is applied. Is it carried out by the agent system designer, or is it a method for an agent to reason at execution?

Reasoning can be either practical or social, or perhaps a mix of the two.

Dynamism is a category to consider how the reasoning can change, to adapt to new situations or roles for the agent.

Type of Interactions describes what interactions the approaches can handle. Is it a method for multi-purpose reasoning about many types of interactions, or is it suited only to a specific task.

Cooperative agent techniques covered methods for agent coordination such as GPGP and Joint Intentions. In general these are theories of agent cooperation designed for use in agent designs. However there are implemented systems, so they cover the spectrum through to applications. The bias is towards the theory though, as that is the main contribution here as far as reasoning is concerned. These approaches tend to be relatively domain independent in that they can be applied to any design – so long as it is a closed, cooperative system. In this respect they are useful only for specific systems. The focus of these theories is purely design-time. They act as aids to the designer wishing for their agents to act in some (statically determined) manner. On the whole these techniques are only concerned with social reasoning, but a slightly different case would be GPGP which defines how the agent's action planning should be coordinated and so impacts upon the practical reasoning domain. The types of interaction are obviously purely cooperative.

Communication based social reasoning covered a range of techniques, where the agent

[1] The inspiration for this comes from the "Description Level Axis" classification assigned to papers for the AAMAS 2009 [IFAAMAS, 2009] conference.

Table 2.1: A Summary of Social Reasoning Techniques. Circles represent the weighting of the approaches along the scale for each category.

Criteria / Technique	**Level of Abstraction** Theory - Application	**Generality** Domain - Specific Independent System	**Focus** Design - Execution Time	**Reasoning** Social - Practical	**Dynamism** Static - Dynamic	**Type of Interactions** Any - Specific Type Task
Cooperative Agent Techniques						
Comms. based Social Reasoning						
Planning with Speech Acts						
Argumentation						
Behaviour Constraints						
Opponent Modelling						

was either reasoning based on communication or reasoning over communication itself in terms of protocols used. The work covered mostly concerns itself with the method used to reason and implemented applications, rather than general theories (though of course nearly all work has a theoretical base –this table and summary is only a rough overview). Reasoning in this case happens both in the design of the agent and at execution. The dynamic protocols of McGinnis [2006] allow for reasoning at design time.

Speech acts are a well established means for incorporating agent communication into planning. As the original goal was to allow for social actions inclusion with practical ones, it falls midway between true social reasoning and practical. The long development of speech acts and their application to not only planning but uses such as institutional control, commitment and mentalistic models of communication mean they have covered the range from theory to application. The focus is nearly always design time and directed at the specific task of communication planning.

Argumentation based approaches to agent reasoning are still relatively new, with few progressing beyond theory into applications. For the arguments to be easily generated the agent system tends to rely on strong assumptions or a specific setup for a style of argumentation. Related to the lack of implementations, negotiation theories can only guide design and so result in static rules for negotiation and tackling specific tasks.

Norms, virtual organisations and other agent behaviour constraints cover quite a variety of approaches from theories to implemented systems. As the concepts are quite general they can apply to many domains and types of interactions. They are labelled with a static design time focus. Although there is work to allow agents to change the institutions they are part of at runtime, for example, the goal of these constraints is to simplify agent reasoning through system design. Allowing only behaviour adhering to specific norms, or certain actions only in specific roles, is primarily for the purposes of constraining agent reasoning (often for simpler coordination).

Opponent modelling techniques in MAS are rare and at the theory level directed at specific modelling tasks. There are some exceptions, such as Finite-State Machine (FSM) based systems which could be considered more general. They are unique in that the key focus is to allow for dynamic reasoning, albeit at specific tasks.

The summary table allows us to make some general observations. Most agent reasoning methods in the literature consider varied levels of abstraction. There is a bias to-

ward theories, as even implementations require sound theories on which to base themselves. The categories covering a wider spectrum are those which are more general and cover more varied and mature work – so this is expected. It seems that most theories are domain independent (even if they only handle specific types of interaction), but this is linked to their level of abstraction. Abstract theories which can only influence an agent's design and say nothing of execution are bound to be more generally applicable than concrete reasoning techniques. Design time reasoning for specific tasks is an overwhelming trend – as naturally this allows assumptions to simplify the complex tasks embodied in agent social reasoning. Dynamic reasoning is rare, perhaps unrelated to the other columns and again a symptom of the difficulty of reasoning agent design and implementation.

Continuing to summarise the review as a whole, key points identified are as follows:

- A lack of generality – much specialisation, certainly in approaches providing practical help to the agent designer.
- Many systems "design-in" social reasoning, where the agent designer does the reasoning and not the agent at run-time.
- There is a lack of implementations, test cases or common problem domains.
- A lack of reasoning based on communication, especially work that is not limited to proofs of a system's properties.
- A lack of dynamic approaches – both to model systems and for agents to reason – where societies change over time.
- The issue of interaction between social and practical reasoning arises – are they even different?

In general, there would appear to be an overall theme that social reasoning is a hard problem and as a result there are many different approaches to common problems. This leads to much specialisation in approach (for example the BDI logics designed to address particular properties) and aiming to fix small, specific problems in other theories. There appears to be a lack of links between the various issues due to this specialisation, a case of not being able to see the forest for the trees. This lack of generality in theories that provide practical help to an agent designer is another weakness, here there is a lesson that can be learnt from the success of BDI. BDI theory is general and can be adapted to many applications, yet still provides enough constraint on how an agent

should reason to be useful. There is a lack of equivalent theories for social reasoning. The drive for a more general approach presents problems however – what level should such an approach be at? Too abstract and it has little practical use, too concrete and the fine details render it opaque to widespread application.

The conclusions drawn here motivate the goals of this thesis, as introduced in Chapter 1. The key issue from the above addressed with the ESB framework is the lack of generality in social reasoning. This is manifest in two ways. There are the differing levels of abstraction present and most approaches tackle different, specific, tasks. The benefits of a general framework can be seen for comparison of social reasoning schemes, integration of differing schemes and agent design. These advantages have been shown in the BDI paradigm, which has guided practical reasoner design and provided a bounded rational general model for implementation. ESB as a general social reasoning framework can realise some of these benefits in this domain.

By considering general social reasoning, by extension this thesis also considers the question of the relationship between social and practical reasoning. The argument that social reasoning could be productively considered separate to practical reasoning is a strong one. This summary has shown most approaches could be considered pure social reasoning, setting aside practical reasoning concerns for a separate process – effectively separating social reasoning. This provides the motivation to consider ESB as a social reasoning process discrete from practical reasoning. The dynamic reasoning of agents has been shown to be an important topic in this goal. Any general framework must allow for change in agent reasoning, as this survey shows that it is either a key part of certain reasoning techniques or conspicuous by its absence. If ESB is to be able to not only cover a variety of reasoning theories but allow for further development of agent reasoning, reasoning about change is surely vital. Designed-in reasoning is also an identified weakness in existing work. The goal of ESB is not to propose a particular reasoning approach, so this weakness is not tackled directly, but a sufficiently general social reasoning approach supporting dynamic reasoning is by necessity very powerful and expressive. This expressiveness could allow for carrying out more reasoning at execution time versus design time. Despite this expressiveness, creating a computational framework was also a key motivation behind this thesis. In conducting this review the woeful lack of implemented social reasoning agents became evident and any work that can be done to bridge this gap will allow important lessons to be learned.

Having summarised social reasoning in MAS and explained the conclusions that lead

to the motivations behind this thesis, the rest of this document is concerned with setting out how the problem of creating and evaluating a general social reasoning system was tackled.

2.2 Related Work

The background section has thus far reviewed the field of social reasoning in MAS, the conclusions that can be drawn and set out the motivation for the goals of this thesis based on this review. The work presented so far is therefore work related to the goals of this thesis and its motivations. This section covers work more closely related to the approach taken to tackle this problem – the ESB framework introduced in section 1.3. As the aim of this thesis is to address a perceived gap in the current state of MAS research, rather than extend existing work, there is little that is directly related. However there is some work which shares motivations, central ideas or tackles parts of the problems faced. This work is referenced here and related to this thesis.

Preceding work with ideas that eventually led to those behind the ESB approach of this thesis is presented by Nickles et al. [2004a]. Although this seems superficially very like ESB, with talk of "expectation networks", in detail it is only loosely related. These expectation networks are solely concerned with communication between agents. They are a means to capture communication patterns probabilistically, dependant on conditions identifying certain states in protocols and the probable next message. The motivation for this approach is that communication is necessary for agents to interact and so any social relationships between agents can be described in terms of the communication structures – expectation networks. This thesis argues that whilst this might be true, there are types of agent reasoning that cannot be captured in this way. To capture and implement an agent's social reasoning the mental attitudes of agents and inferred states of others must be considered and updated. This at its most basic requires revision of beliefs, based on the observations of others and it is a general framework for this that ESB aims to provide. In this respect, the work in Nickles et al. [2004a] could be seen as a far more specific form of social reasoning concerned only with identification of communication protocols in use.

Sindlar et al. [2009] discuss abduction of other agents' mental states with a view to creating more believable interactions amongst virtual characters. An assumption of

the ESB approach is that some model of another agent's reasoning is required to reason (socially) about them for the purposes of interaction. That is the same conclusion in this work. Their motivation differs, however. They aim to develop a means for agents to infer mental states of others to interact socially in believable ways. They do not define how these inferred beliefs should be used by the agent's reasoning, though they do note that "Abduced explanations would have to be incorporated into these mental models, requiring a revision mechanism and preservation of integrity constraints". Additionally, ESB does not proscribe any specific mechanism for an agent to reason, it is up to the social reasoning scheme being implemented. This need for belief revision as a component of a social reasoning system was a key concern in the design of ESB. At its core it could be described as a generic mechanism to handle this required belief revision based on observations of others.

There exists work that is motivated by similar observations of the state of MAS research. One example is the work of Dennis et al. [2008]. The aim of this work is to create an intermediate agent description language, with the goal of being semantically equivalent to the common BDI agent languages. This is somewhat different to the goals of this thesis, but they have a common motivation. They observe that there are many different BDI implementations and yet they all have common properties and a common representation might allow easier comparison between approaches and transfer of concepts. This goal is certainly similar to that of this thesis – which tackles the same problem as far as it concerns social reasoning, albeit at a higher level of abstraction. This is a reflection on the maturity of the relative areas of research. Practical agent reasoning systems have been in development for decades and as such there are a rich variety of implemented languages to create agents. This is not the case with social reasoning. Whilst there is a body of theory on individual techniques, there is no unifying theory or architecture, or common bridge from theories to implementation. This observation of the current state of social reasoning research was a motivation for this thesis.

The work presented here is not the only MAS research to describe a reasoning in terms of social expectations. Cranefield [2007] describes a different concept for social expectation, introduced in [Cranefield, 2006]. This work is based on the hyMITL$^{\pm}$ logic (a hybrid Metric Interval Temporal Logic) and its use to capture social expectations such as norms and commitments in electronic institutions (EI). Their goal is to build a monitoring tool with this logic, to enable the creation of a compliance monitor for

norms and their like in EIs. Like ESB, there is no explicit concept of commitment or norms built into the logic, but rather the system presented is argued to be capable of capturing the relevant properties. Due to the focus on detecting violations of expectations, this is in terms of conditions that must be met at certain times (hence the focus on time in their logic). The structure of an expectation in this context is also similar to ESB. An expectation is held under some condition and then monitored until it is fulfilled or violated according to some other specification. In contrast to ESB, the social expectations here do not say anything about how an agent should reason and how the interaction of an agent's social and practical reasoning should be managed. The aim is not to capture an agent's social reasoning method, but rather to monitor properties of interactions at a system level. This underlying distinction in the goal of ESB compared to hyMITL$^{\pm}$ is the main difference between the approaches. The approach taken for implementation also differs. hyMITL$^{\pm}$ aims for logic specification and checking of properties though partial evaluation of expectation rules, whereas ESB tends toward declarative specification of an agent's reasoning process to allow for execution.

F. Dignum *et al.* have also tackled the problem of integrating social reasoning with practical reasoning, albeit in a more specific manner than the general approach of ESB. Their B-DOING architecture is described in [Dignum et al., 2002b,a, 2000]. Their approach is to extend BDI to include the different motivations for an agent's actions to include not only beliefs, but desires, obligations and norms. A logical system is presented to describe the semantics of their approach and how these concepts may be captured and integrated by way of preference orderings over goals. In contrast to the meta-level ESB reasoning, the object-level reasoning of B-DOING agents works on the assertion that these basic motivations (or social objects) for action are sufficient to capture an agent's social concerns. The focus in their work is also more toward the logical representation of agent reasoning than the declarative specification based concerns of the ESB approach in this thesis – which arise from a desire to bridge the gap from theory to implementation.

Later work in van der Vecht et al. [2009] is also comparable to the approach of this thesis, as they aim to capture social concepts for agent reasoning in a separate process to practical reasoning. This is close to the central concept of the work presented here, only the approach is different even if the motivation is similar. Here meta-reasoning outside of the practical reasoner is used to process events (observations, such as communication) and thus influence practical action decisions. The goal of this processing

is to determine what sort of reasoning is appropriate and how much reasoning, for a particular situation. This is similar to the goal of bounding social reasoning that ESB has, only limited to heuristics applied to an agent's observations. In practice this looks to be mostly useful for processing communications to allow agents to reason more effectively, but the published work is at an early stage.

As mentioned at the start of this section, there is no research directly related to this thesis. It does not directly extend existing work. This section has therefore presented an overview of material that whilst similar in some key respect is also fundamentally different in either execution, motivation or goals. There is of course a large body of MAS work that touches on that presented here in some small way, so this section is confined to include only the most obviously related research that has served as some influence on the work presented in this thesis.

2.3 Conclusions

This chapter had two main objectives. The first was to provide a general background to the many varied types of social reasoning in MAS, thus setting the scene for the general framework developed for this thesis. Section 2.1 provided this overview, with section 2.2 going into more detail to cover literature more closely related to the approach of this thesis. The second objective was the motivation of the main goals of this thesis. These goals arose from observations of the weaknesses of existing social reasoning approaches and so the motivation came after the general background in the summary of section 2.1.3. The related work also obviously to some extent motivates the development of ESB. As described in the previous section there is no related work targeting the same problem, so overcoming this shortfall is an important goal.

Throughout this chapter the lack of generality in existing social reasoning systems has been stressed. This observation was the key inspiration to create a general social reasoner. The decision to pursue a discrete social reasoner rather than extend an existing practical reasoner was also borne from this review. Principally it was from the observation that most social reasoning is considered separately and yet there are common themes to the approaches in the way agent beliefs are managed and reasoning changed on observation and expected mental states of other agents. This decision also allows a further benefit with regard to this thesis – the potential for easier implementation of

social reasoning techniques. This could arise from the general framwork giving structure to an agent design and allowing easier integration with existing practical reasoning engines.

So to summarise, this chapter has provided the motivation behind the goals of this thesis and the main problems it aims to address. In the process the relevant background material has been introduced, providing an overview of agent social reasoning. Of course there is other literature that is relevant to part of the ESB solution created. This will be introduced where relevant as it is not background material, rather specific techniques that have been utilised, or reasoning schemes referenced.

Chapter 3

The Expectation-Strategy-Behaviour Framework

The first chapter introduced the problem tackled in this thesis, that of creating a general social reasoning framework separate from practical reasoning. This was motivated and situated in existing work by the background chapter and now this chapter will provide an overview of the ESB (Expectation Strategy Behaviour) framework that has been developed. This starts with a high level overview, before moving on to an example that can be referred to throughout the chapter. The focus is on ESB at the theoretical level, introducing the various definitions and semantics that define an ESB specification on social reasoning. This is then extended with the algorithms used to process these specifications, illustrating how the ESB reasoning process is carried out. Some of the first sections are based on the previously published paper [Wallace and Rovatsos, 2009], which provides an overview of ESB.

3.1 An Overview of ESB

In general, the internal state of agents is usually not observable to other agents, e.g. in systems in which agents pertain to different users unwilling to reveal all details of their agent's internal design. As such, social reasoning mostly concerns hidden properties of the system, i.e. beliefs about non-observable features of the system such as other agents' beliefs and goals, their trustworthiness, etc. This is a property that can be seen in nearly all the approaches described in the background chapter.

From an agent design point of view, specifying what an agent should think about others in specific circumstances requires the specification of reasoning rules that define how beliefs about these *non-observable* properties will be formed and updated based on *observable* features of the overall behaviour of the system (such as environmental state changes, agent actions, etc).

What is needed are belief revision mechanisms that work in practice for the agent when interacting with others and in accordance with its local beliefs, its objectives, and its deliberation and planning mechanisms. Such belief revision strategies would normally be implicit within the belief update functions of the architecture. These functions form part of the sense-reason-act cycle of any architecture suitable for use in agent-based systems.

The ESB framework realises several benefits through separating out social reasoning based belief update, for example:

1. It simplifies the agent implementation process from the designer's point of view by providing automated support specific to reasoning patterns that are common in social reasoning.

2. It provides a modular way to specify and design social reasoning schemes, allowing easy extension.

3. It enables application of *bounded rationality* principles to social reasoning (in a similar way as these are applied to general practical reasoning in architectures such as BDI).

ESB is fundamentally based on the notion of *expectations* as beliefs regarding hidden properties of the system. Expectations contain a specification of a "test" that will be used to verify whether the property held or not. They also contain "responses" which describe how the agent will update its beliefs depending on the outcome of the test (where different expectations in the agent's overall "expectation base" may affect each other). Expectations, together with *strategies* (which describe the style of reasoning that the agent applies to its expectation base) and *behaviours* (that capture how the currently held expectations affect actual agent behaviour) provide a natural way for describing concrete social reasoning methods and for devising modular social reasoning designs.

It should be emphasised that ESB does *not* propose a concrete social reasoning method.

Instead, ESB is based on the very generic idea of "managing belief revision strategies about hidden properties in a boundedly rational but socially meaningful way" that could be applied to any beliefs concerning social reasoning concepts like those listed above. In this way, it aims to achieve a level of generality similar to that of "meta-architectures" like BDI, while at the same time capturing all the elements that are frequently used by social reasoning schemes to support the process of their design.

The ESB framework is an abstract model for practical social reasoning systems based on the concept of *expectations*, which can be defined as follows:

> An *expectation* is a *conditional belief* regarding a statement whose truth status may eventually be *verified* by a *test* and *reacted* upon by the agent who holds it.

More concretely (and as will be defined more formally below), it is assumed that an expectation concerns a belief Φ held by an agent A under a condition C. Depending on the outcome of a test T, the agent will apply response ρ^+ if the expected belief was confirmed (positive response), and ρ^- if not (negative response).

Strategies specify ways in which sets of expectations are processed. The easiest way to conceptualise this process is to think of all possible sets of expectations that could arise from future observations of test outcomes and to think of a strategy as a particular way of traversing the graph that results from mapping out all possible future expectation-relevant events. Simple strategies include bounding the depth of predictions, or excluding expectation sets that are unreachable from the initial state of the agent at execution time, or impossible because of mutually exclusive test outcome. More complex and truly "social" ones include, for example, assuming that different states have different utilities and applying game-theoretic reasoning in the exploration of future expectation states.

The purpose of strategies is to determine what model of belief revision the agent will use to evaluate the conditions of *behaviours*. Generally speaking, behaviours are rules that determine how the state of the expectation base affects the reasoning agent's actions. They are conditioned on statements about expectations and the truth value of these will be established applying the agent's strategy to the expectation base.

In itself, ESB does not of course generate any concrete agent behaviour and has to be integrated with some more general reasoning and execution architecture to yield a complete agent design. Beliefs are a universal component of practical reasoning

architectures and beliefs can provide a mechanism for the separate ESB social reasoner to influence an agent's actions in the general case.

So for the purposes of explaining the ESB theory, behaviours always have an overall belief change in the agent as their only consequence, i.e. each ESB behaviour is of the form "**if** ψ **then** (don't) believe β" where ψ is a condition on the expectation base. This means that a behaviour simply causes a belief update and the assumption is that an architecture such as BDI will be affected by this update appropriately, e.g. by making certain plans available or unavailable depending on the truth status of β. The examples all assume a belief-based practical reasoning architecture, but this is not necessary. For example, if ESB was to be coupled with a reactive agent design, β could be replaced by a real physical action.

Figure 3.1 shows an overview of the complete ESB framework and helps provide an intuition as to how the complete system works. The agent design is specified in terms of the expectations, strategy and behaviour rules. ESB provides general algorithms to create the expectation and strategy graphs that are used to allow complex behaviour conditions to be checked. These behaviours then influence a practical reasoner component by way of beliefs – and it is here the interaction with the environment is handled.

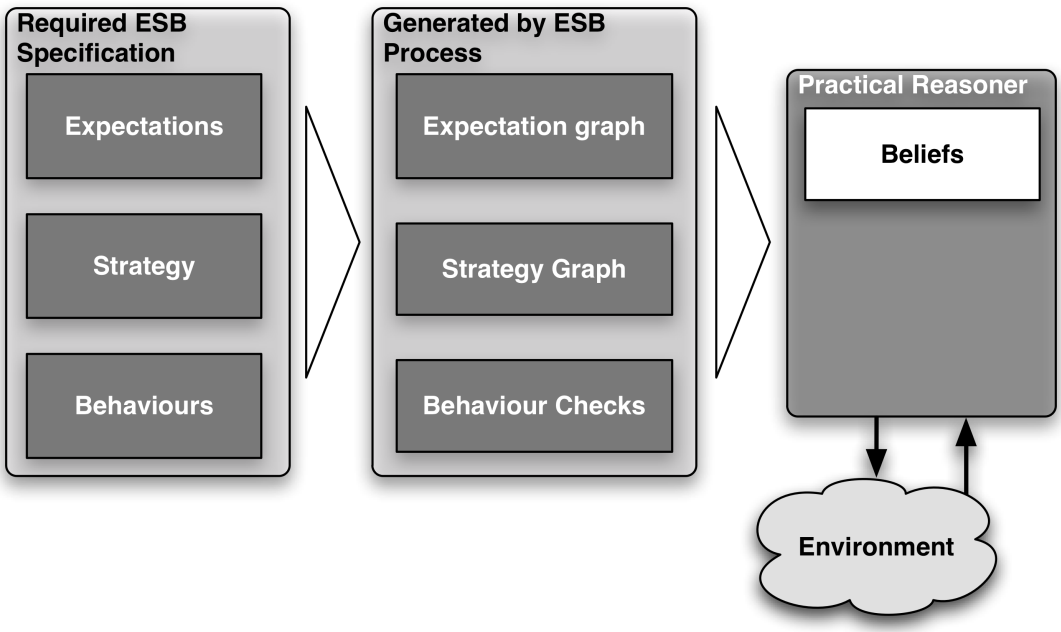

Figure 3.1: Overview of the ESB Framework .

3.2 Introducing an Example

An in-depth example is useful to provide an overview of ESB. In this section the Rummy [Rummy.com] example from the introduction will be repeated and extended. Card games, particularly Rummy, can provide a nice simple example domain as the strategies that the opponents might use are known (for simple games) but dependant on the cards they hold – which are unknown. This is a simple and intuitive case where an agent must base its actions upon reasoning about another agent's unobservable mental processes.

For the purposes of this example it is only important to understand that players are trying to collect either runs of cards in a suit, e.g. {2♣,3♣,4♣}, or sets, e.g. {2♣,2♢,2♡} (given a card X, other cards that could be part of a set/run with that card are referred to as a "member of X's sets/runs"). *Melds*, as these sets or runs are called, must contain a minimum of three cards. On their turn a player must either pick up a card from the deck, which is unseen, or from the top of the discard pile, which is visible to all. They then create any melds they can and place a card on the top of the discard pile, play then passes to the next player.

In terms of ESB, assumptions (Φ) must be made about these cards based on what the opponent has picked up and/or discarded in the past (C). These assumptions are borne out (or not) by the opponent's future plays (T) and the reasoning rules can be adjusted (ρ^+, ρ^-). Behaviour can then be chosen based not only on current expectations, but also on potential future melds they may collect – accounting for an opponents possible change in reasoning in the future.

3.2.1 The Example Scenario

The example involves only two players, where the ESB agent A reasons about its opponent B picking up a 2♢. The example considers only relatively naive reasoning on A's part: that B will either be collecting a run of diamonds around 2, or a set of 2s (A assumes B won't hedge her bets by collecting cards for both a run and a set). The reasoning for this example is encoded in the expectations in Table 3.1. As there are limited actions available to the players and both tests and conditions depend on observations of these actions, the T and C components appear similar in the Rummy example, this need not be true in the general case.

Table 3.1: Expectations in the Rummy example: the initial expectations (1,2,5) are shown in bold face; tests are broken down into $+(\ldots)$ and $-(\ldots)$ events that would confirm/disappoint the expectation (table reproduced from Wallace and Rovatsos [2009]).

N	C	Φ	T	ρ^+	ρ^-
1	B **picked up 2♢**	B **is collecting 2s**	**+(**B **picks up a 2)** **¬(**B **ignores or discards a 2)**	**remove({2,5})** **add({3,4})**	**remove({1})**
2	B **picked up 2♢**	B **is collecting ♢ run**	**+(**B **picks up member of 2♢ run)** **¬(**B **ignores/discards member of 2♢ run)**	**remove({1,3,4})** **add({5})**	**remove({2,5})** **add({3,4})**
3	B discarded 2♣	B not collecting 2s	+(B ignores a 2) ¬(B picks up a 2)	–	remove({1,3}) add({2,5})
4	B ignored a 2	B not after 2s for run or set	+(B ignores a 2) ¬(B picks up a 2)	–	remove({1,3}) add({2,5})
5	B **picked up 3♢**	B **is collecting ♢ run**	**+(**B **picks up another member of 3♢ run)** **¬(**B **ignores/discards member of 3♢ run)**	**remove({1,3,4})** **add({2})**	**remove({2,5})** **add({1,3,4})**

3.2.1.1 Expectations

Below is the more intuitive natural language reasoning behind each expectation rule in Table 3.1:

1. When the opponent picks up a 2♢, hold the expectation that they are collecting a set of 2s. The test is if they pickup another 2 or complete a set. If they ignore or discard a 2, then the expectation was incorrect.

2. When the opponent picks up a 2♢ expect them to collect a run of diamonds. This is reinforced by them completing the run or picking up another member of the run. Conversely, if they ignore or discard a member card the expectation is false.

3. If a 2♣ is discarded then expect that they're not collecting a set of 2s. Ignoring a 2 confirms this, picking one up denies it.

4. If a 2 is ignored, then expect they've no need for 2s for a run or set. The test is as above, ignoring or picking a 2.

5. If a 3♢ is picked up then expect them collect a run of diamonds. This is reinforced by them completing the run or picking up another member of the run. Conversely, if they ignore or discard a member card the expectation is false.

The responses take the form of adding and/or removing the specified expectations as indicated, depending on the test outcome. The initial set of expectations held by the agent are $\{1,2,5\}$ This represents an agent with the initial position to be open to the notion of the opponent carrying out either strategy. As the purpose of this example is to illustrate the workings of ESB, the rules have been restricted to specific examples for simplicity. For example, it should be obvious that 2 and 5 are both instances of some more general rule – to expect the opponent is collecting a run when they pickup one member card of it.

The expectation graph is a key intuition to understanding ESB and how behaviour conditions on the expectations are evaluated. The graph provides a representation of how the agents expectations are modified based on observation of T test results. Figure 3.2 shows the accessible expectation graph generated from the expectations in Table 3.1 and initial state S2. Vertices are labelled with the expected belief Φ of each expectation tuple, as this makes consideration of the relevant behaviours easier. The initial state S2

is indicated by the double lines. Edges represent transitions specified by the responses. A loopback to a state indicates that the ρ^+, ρ^- responses of some expectations do not cause a change in state.

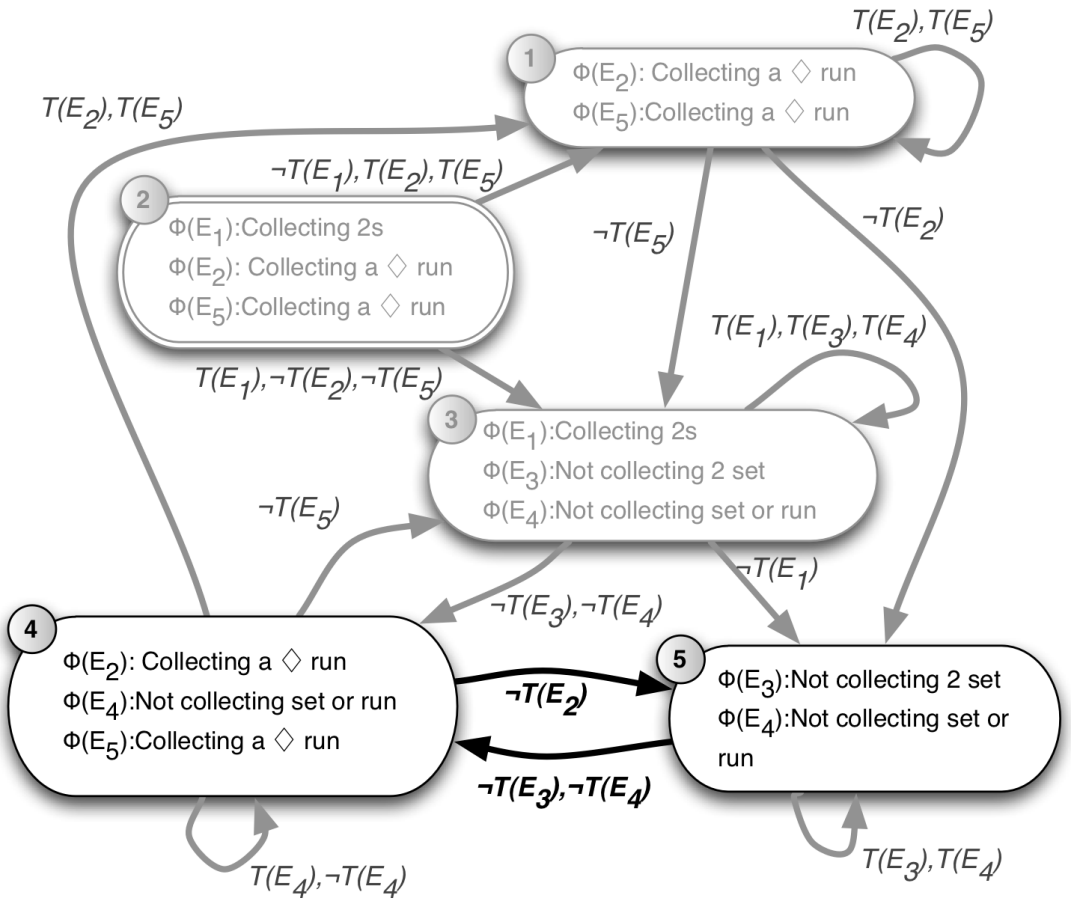

Figure 3.2: Accessible Expectation Graph, showing the states reachable from the initial state S2 .

3.2.1.2 Strategies

If we assume that the game is expected to end soon, this presents a possible strategy for the example agent. For this reason, *A* has decided not to consider all possible future strategies, but to only reason about what *B*'s strategy might be at this time.

A simple strategy that can be applied in this case is to only consider the expectation graph to a depth of one from the current state and to ignore loops. The justification of this strategy is that responses are triggered by observations (which update the status of tests) and these must come from observing cards being played or picked up. An

example of this strategy is shown by the bold states in the example of Figure 3.2, where the strategy is applied to the graph assuming a current state of S5.

3.2.1.3 Behaviours

Behaviours take the form of conditions (on the strategy graph and expectations) and actions (which may be direct actions or internal ones). For the purposes of this example, the agent cares only which cards are safe to discard based on the model of the other player's reasoning. So three simple behaviours are as follows:

1. Only discard a 2 if B is not collecting 2s.
2. Only discard a 3 if B is not collecting 3s.
3. Only discard cards B is not expected to be collecting in the future.

Whilst these make sense intuitively, it is worth translating them into specific conditions on the expectations, as this will prove useful for the rest of the example:

1. (a) Discard 2♢ if $\neg\Phi(E_1) \wedge \neg\Phi(E_2) \wedge \neg\Phi(E_5)$
   (b) Discard 2♣ if $\neg\Phi(E_1)$
2. Discard 3♢ if $\neg\Phi(E_2) \wedge \neg\Phi(E_5)$
3. (a) Discard 2♢ if in all accessible states $\neg\Phi(E_1) \wedge \neg\Phi(E_2) \wedge \neg\Phi(E_5)$
   (b) Discard 2♣ if in all accessible states $\neg\Phi(E_1)$
   (c) Discard 3♢ if in all accessible states $\neg\Phi(E_2) \wedge \neg\Phi(E_5)$

Here, "in all accessible states" refers to all states accessible from the current expectation state (assuming all $\Phi(E_i)$ are true), according to the strategy. So in the case of this example, it means all vertices adjacent to the one corresponding to the current state.

The first two are simple and refer to the current expectations, those the agent reasons with in the current state whose conditions apply. The third is included as an example of a behaviour which depends not just on current reasoning, but on future reasoning. This requires consideration of the expectation graph (and its restriction, the strategy graph).

3.2.2 Working Through The Example

The "end product" of the ESB system is the behaviours, so it is important to consider which ones apply and how they influence the agent's actions. For the purposes of explanation consider the scenario where the agent holds a 2♣ and 3♢ and must make the choice of which to discard based on applicable behaviours. Initially we will consider the complete expectation graph with no strategy applied.

3.2.2.1 Agent is in State 2

This is the initial state the agent finds itself in at the start of execution. Depending on observations, expectations that the opponent is collecting both a run and 2s might hold, so it is not safe to play either card. Even if none of these currently hold, the third behaviour not to play a card the opponent may desire in the future also holds. This is because there are states reachable from state 2 where it is possible to expect them to be collecting both the 2♣ and the 3♢.

In fact, this is the case for any state the agent might be in. Behaviour 3 always holds as far as it applies to both 2♢ and 3♢.

3.2.2.2 Agent is in State 5 and a Strategy Applies

Now we consider the effect a strategy can have. As mentioned earlier, if we assume the end of the game draws near, then we can apply the strategy of only considering the expectation graph to a depth of 1. This yields the strategy graph as highlighted in figure 3.2. We might expect the opponent is collecting a run and so we cannot discard the 3♢, however we do not expect them to be collecting a set so we can safely discard the 2♣

3.3 A Formal Definition of ESB

Having introduced ESB through an extended example, this section now moves on to define what an expectation is. This is the core of what ESB is and a major part of the work presented here. Defining what is meant by expectations, strategies and the

behaviours is necessary for several reasons. From a theory design point of view it allowed for clear discussion of the pros and cons of various definitions and implications of different semantics to be explored. The motivation behind creating this logic-based definition was the goal of implemented social reasoning agents. A big part of the motivation behind tackling the problem of separating social reasoning from practical was the perceived need to tackle the gap between agent theory and implementation. There are many social reasoning schemes and yet comparatively few implemented reasoning agents. Correspondingly, a means to capture and *implement* social reasoning more generally is the main goal of this thesis. To create an implementation of a theoretical framework it is necessary to know what exactly is meant by an expectation and how it is updated. This is the purpose of the formal definition presented in this section.

Often agent reasoning is defined formally with a view to proving that it meets some properties, certain things can be expressed or compared to existing formalisms in some way. This is not the goal here. Whilst purely formal approaches to explore and evaluate agent reasoning are valid and fruitful areas of research, it was simply not the purpose here, tackling the theory-implementation divide present in current MAS research was. The formal definition is as much as is necessary toward this goal.

3.3.1 A Logic for Expectations

The exact choice of a language to express the ESB framework was not in itself crucial. The goal was to represent ESB and provide a precise way to specify what was meant by an expectation rather than verify any particular formal properties, or rely on any properties of the formalism used. The main concern was that the logic was able to fully capture what an expectation is and how it changes. This implies a logic with a notion of time, actions and agent belief, as the interface between social and practical reasoning is beliefs at the most basic level.

$\mathcal{LORA}$ – Logic Of Rational Agents – is a multi-modal logic described by Wooldridge [2000] which takes as inspiration other BDI logics. It is a multi-modal propositional BDI logic which combines modalities for belief, desire and intention with aspects of dynamic logic and CTL temporal logic, using standard propositional notation. The reasons for choosing this logic for specification are simple, it is very expressive and there is a good book dedicated to it [Wooldridge, 2000]. Indeed it is possibly an over-expressive logic as it presents very few barriers indeed as to what can be represented.

This was a benefit during the design of ESB however, as it allowed many ideas to be tried out.

3.3.1.1 An Overview of $\mathcal{LORA}$

Before getting to the more detailed description of ESB, a brief overview of $\mathcal{LORA}$ is appropriate. This section assumes some familiarity with modal logic in general and BDI logic. $\mathcal{LORA}$ is a multi-modal logic best described as first-order branching time logic (CTL*, Gabbay et al. [2000]), combined with additional modalities to represent the beliefs, desires and intentions of agents and dynamic logic to consider their actions.

The temporal logic component of $\mathcal{LORA}$ takes a CTL* branching view of time, with only one past, as represented in Figure 3.3. Formulae in particular states are known as state formulae and those referring to one possible future path, path-formulae. Below in Table 3.2 are the various path connectives and their informal meaning.

Table 3.2: Path Connectives

$\bigcirc\varphi$	In the next state, φ is true.
$\Diamond\varphi$	Eventually – in some future state φ is true.
$\Box\varphi$	Always – in the current state and all others on this path φ is true.
$\varphi\ \mathcal{U}\ \psi$	φ is true until ψ is true (and ψ will eventually hold)
$\varphi\ \mathcal{W}\ \psi$	Weak until – φ is true until ψ is true (ψ may not eventually hold)

The two path quantifiers are: **A**, on all future paths and **E** on some future path. By combining these with the path connectives, properties such as "φ is inevitable" can be expressed as $\mathbf{A}\Diamond\varphi$.

The BDI component follows the intention logic of Cohen and Levesque [1990]. The modal connectives for these aspects of an agent's mental state take the following forms:

(**Bel** i φ)	i believes φ
(**Des** i φ)	i desires φ
(**Int** i φ)	i intends φ

In the above table i is a constant to denote the agent and φ is a $\mathcal{LORA}$ formula.

An important consideration of modal logics such as these are the axioms used to define them. In this case, belief is KD45 (serial, transitive and Euclidean) and desire and

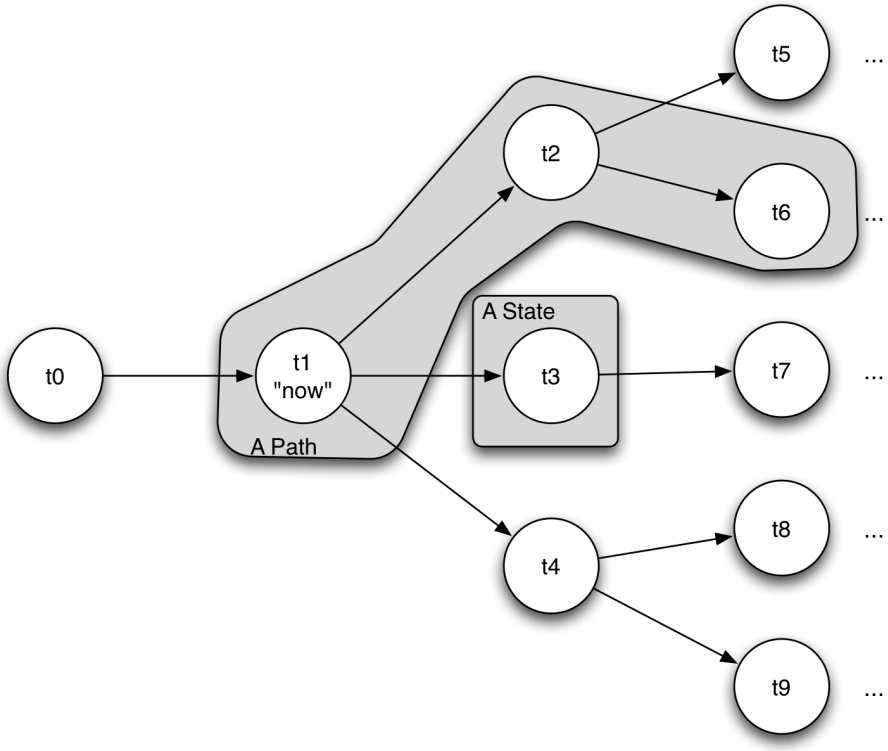

Figure 3.3: The branching time structure, showing a state and a path.

intention both KD (serial). To informally express these properties:

K - If an agent believes φ and that φ ⇒ ψ then it believes ψ.

D - An agent does not believe a contradiction.

4 - An agent believes that it believes φ, if it believes φ.

5 - If an agent does not believe φ, it believes it doesn't believe φ.

$\mathcal{LORA}$ also has a dynamic logic component, to represent the actions agents make, which cause transitions between temporal states. In terms of the temporal possible-worlds model (as above in Figure 3.3), the actions label the transitions and one action may label more than one transition i.e. they are non-deterministic. This dynamic logic component is not used here, so for further details the reader is referred to [Wooldridge, 2000].

The final specification of ESB does not make use of all this language, but this brief overview serves to illustrate the expressive power of $\mathcal{LORA}$ and reinforces the assertion that it is general enough for the purposes of this thesis.

3.3.2 Expectations

With a language to express semantics of an expectation, we can now start to define expectations more precisely. The expectation "whenever condition C holds, agent A will expect Φ, check it with test T and react with ρ^+ if it is true or ρ^- otherwise" is represented as:

$$\mathbf{Exp}(N\ A\ C\ \Phi\ T\ \rho^+\ \rho^-) \tag{3.1}$$

where

N is the *name* of the expectation, taken from a set of expectation labels $\{N, N', \ldots\}$,

A is the *agent* holding this expectation, taken from a set of agent names $\{A, A', \ldots\}$,

C is the *condition* under which the expectation holds,

Φ is the expected belief, i.e. an event or an expectation[1] that another agent is assumed to hold, or some other inferred belief about the world.

T is the *test* which confirms (or rejects) the expectation of Φ (an observable event),

ρ^+/ρ^- are the positive/negative *responses* to the test T; these are modifications to its internal state, modifying the set of expectations it holds defined in terms of expectations to add or remove.

Now using $\mathcal{LORA}$, we can define the semantics of expectation tuples and how they change with observations as follows:

$$\mathbf{Exp}(N\,A\,C\,\Phi\,T\,\rho^+\,\rho^-) := \mathbf{A}\Box(\mathit{current}\ \mathcal{W}\ \mathit{update}) \tag{3.2}$$

where

$$\begin{aligned} \mathit{current} &= (\mathbf{Bel}\ A\ C) \Rightarrow (\mathbf{Bel}\ A\ \Phi) \\ \mathit{update} &= \Big((\mathbf{Bel}\ A\ T) \wedge \mathbf{A} \bigcirc \rho^+\Big) \vee \\ &\qquad \Big((\mathbf{Bel}\ A\ \neg T) \wedge \mathbf{A} \bigcirc \rho^-\Big) \end{aligned}$$

and

$$\begin{aligned} \rho^+ &\equiv \mathbf{Exp}(N'\ A'\ C'\ \Phi'\ T'\ \rho'^+\ \rho'^-) \wedge \ldots \\ \rho^- &\equiv \mathbf{Exp}(N''\ A''\ C''\ \Phi''\ T''\ \rho''^+\ \rho''^-) \wedge \ldots \end{aligned}$$

[1] Nesting expectations allows modelling of other agent's mental states and adaptive behaviour on the part of either agent. The extension required to the formalism is covered later in section 3.5.

such that the $\rho^{+,-}$ responses are conjunctions of expectations.

Of course, expectations, or parts thereof can be empty or simply "true". Informally, the above equation can be described as follows:

> *In all paths it is always the case that: Belief in C implies belief in* Φ*, until T is believed and in all paths at the next time step* ρ^+*, or* $\neg T$ *is believed and* ρ^- *holds.*

The responses are defined as a conjunction of expectations for one main reason – to keep an expectation after the test is completed. Once the second part of the "until" holds, (**Bel** A C) $\Rightarrow$ (**Bel** A Φ) is no longer guaranteed to. So if we wish the response to have no effect then the response needs to include the current expectation. And if we wish to add another expectation as a response, there will need to be two in the response. Removing an expectation then just involves leaving it out of the response specification.

A point to note about this formalisation is that the "until" structure defining when the expectation is current assumes that it is rational for an agent to not hold a belief about the truth value of T at all. If expectations were not defined this way, an agent would never believe Φ, as it would always believe T or $\neg T$.

If this is not desirable, then the obvious change would be to define separate beliefs for a positive and negative test outcome, and then the agent would be irrational or inconsistent if it held both – a property of an implementation or set of expectation rules that could be checked. This is how the reasoner implementation described in a later chapter works, with discrete positive and negative test cases, and consistency checking left to the agent designer.

To refer to the various parts of an expectation and the various different expectations an agent may hold, we introduce the notation $X(E_i)$ to refer to component X of expectation i in a set of expectations. So, for example, $C(E_2)$ refers to the condition of expectation 2.

As well as the definition of an individual expectation it is also necessary to define the various sets of expectations the agent reasons over. First, the set of all expectations:

$$\mathbf{EXP} = \{E_1, E_2 \ldots E_N\} \tag{3.3}$$

EXP is the total set of all possible expectations a particular agent might have, which is taken to be finite. This restriction has certain implications. For example, responses that

increment an unbounded counter cannot be included. This means all expectations that are possible for *one particular agent* to hold, not the set of all possible expectations that can be expressed using the syntax introduced.

The following two subsets of **EXP** are also used:

- At any time an agent holds a certain number of so-called *active* expectations i.e. exactly those expectations for which $(\textbf{Bel}\ A\ C) \Rightarrow (\textbf{Bel}\ A\ \Phi)$. These form the set $EXP_A \subseteq \textbf{EXP}$.
- An expectation depends on a condition C, under which it holds. The set of expectations whose conditions are currently true, i.e. those for which $((\textbf{Bel}\ A\ C) \Rightarrow (\textbf{Bel}\ A\ \Phi)) \wedge (\textbf{Bel}\ A\ C)$ holds, form the *current* expectation set $EXP_C \subseteq EXP_A$.

The set of active expectations changes according to the results of tests and responses as defined in the expectations themselves. Responses need only add and remove expectations (as this is equivalent to making changes in expectations, if the total set is finite), and below we will use $add(S)$ and $remove(S)$ as the only two possible responses in an expectation which correspond to removing the set S from the set of currently held expectations.

The concept of an agent's *initial state* is sometimes useful for discussion. This is the set of active expectations EXP_A that the agent finds itself in at the start of execution (some of which may or may not be current).

3.3.3 Strategies

Next, in order to define strategies formally, consider the set of active expectations $EXP_A \in \wp(\textbf{EXP})$ which is changed by the responses, whenever test outcomes become known. So the responses can be thought of as defining a relation between the states that the agent's reasoning can be in:

$$xRy \textit{ where } x,y \in \wp(\textbf{EXP})$$

The agent can then be in one of a finite number of expectation states which correspond to possible active expectation sets which are in the powerset of all expectations. These states and the relation between them can be represented intuitively by the expectation graph. More useful still is a restricted sub-graph, containing only those vertices accessible from the initial state, this is the accessible expectation graph. As the initial state

defines which expectations are active at the start of execution with the set EXP_A, it also defines a particular state in the expectation graph. An example accessible expectation graph has been shown previously, in Figure 3.2.

3.3.3.1 Defining the Complete Expectation Graph

The expectation graph is fundamentally a simple structure. The vertices represent all possible sets of expectations. So the set of vertices is the powerset of all expectations:

$$V = \wp(\mathbf{EXP}) \tag{3.4}$$

Defining the edges requires a longer description. The set of edges, where an edge is directed between two vertices (states) u and v is as follows:

$$E = \{e = (u,v) \mid u,v \in V\} \tag{3.5}$$

where

$$\forall\, Expectation_y \in u$$
$$v = (u \cup add\_\rho^+(Expectation_y)) \setminus rem\_\rho^+(Expectation_y)$$

OR

$$\forall\, Expectation_x \in u$$
$$v = (u \cup add\_\rho^-(Expectation_x)) \setminus rem\_\rho^-(Expectation_x)$$

where

$$add\_\rho^+(Expectation_y) = set\ of\ expectations\ added\ by\ \rho^+(Expectation_y)$$
$$rem\_\rho^+(Expectation_y) = set\ of\ expectations\ removed\ by\ \rho^+(Expectation_y)$$
$$add\_\rho^-(Expectation_y) = set\ of\ expectations\ added\ by\ \rho^-(Expectation_y)$$
$$rem\_\rho^-(Expectation_y) = set\ of\ expectations\ removed\ by\ \rho^-(Expectation_y)$$

This can be read as:

> The set of edges E consists of edges e linking vertex u to v (where u,v represent sets of expectations in V) *iff*: For all expectations Y in state u, v is the vertex containing the expectations in u unioned with those added by $\rho+$ and no expectations removed by $\rho+$. This is the case for an arc in the expectation graph that is present from a positive response. Or there is an edge u,v added by the negative response of an expectation. In this case, for all expectations X in state u, v is the vertex containing the expectations in u unioned with those added by $\rho-$ and no expectations removed by $\rho-$.

For an illustration of a complete expectation graph, a simpler example is needed. This is as the number of states in the previous five expectation example would make for a cumbersome graph to present in the constraints of a thesis format. With five expectations there are 2^5 states, as each possible active set of expectations is a member of the powerset of all expectations.

For the purposes of building an expectation graph, only the responses matter, so consider the following simple set of generic expectations:

Name	ρ^+	ρ^-
E_1	null	delete(E_1)
E_2	add(E_3)	delete(E_1)
E_3	null	null

These expectations and responses can be expanded into the graph in Figure 3.4. Although the graph is defined by the responses, the edges are labelled with tests, as it is the test results that trigger the corresponding transitions. Some edges are labelled with more than one test to save space – these represent two or more separate transitions. Of note in this example is the fact that it is not a connected graph – so if the agent is in one of several active expectation states, the others are unreachable – analysis of this sort could help a designer identify redundant expectations or undesired initial states.

With this structure in mind, a strategy is a way to restrict this graph in some way, according to some style of reasoning. This is useful as behaviours rely on checking conditions on the accessible portion of the graph. A strategy formally is therefore a subset of the transition relation E between edges in the expectation graph (described in Equation 3.5).

$$xSy \text{ where } x,y \in \wp(\mathbf{EXP}),\ S \subset E$$

How the relation is restricted is purposefully not specified. This is a concern for an ESB implementation, but could be in terms of forbidden states, sequences of states, transitions etc. In principal any graph operation could be used.

Any way that this graph can be reduced will help to reduce the cost of checking behaviour conditions, or possibly provide a chosen upper limit, depending on strategy. For example, a strategy for an optimistic agent could be to only consider the positive responses of expectations. By reducing the possible transitions, the accessible portions of the graph are restricted.

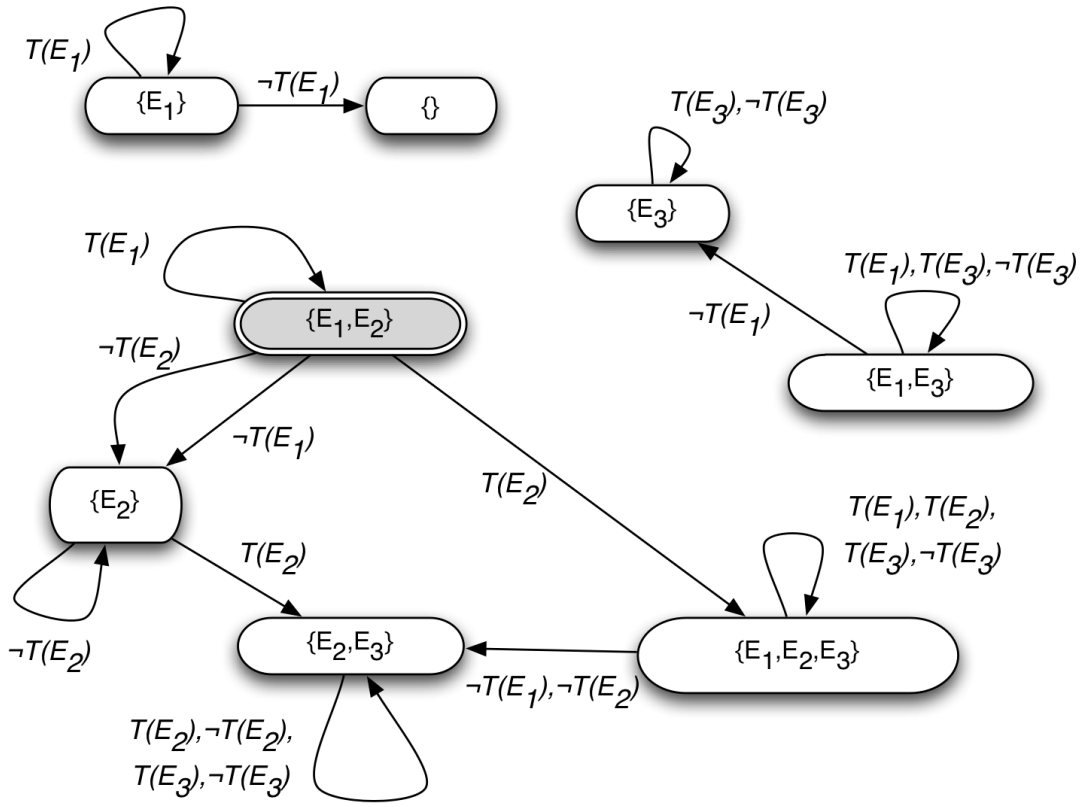

Figure 3.4: Example Complete Expectation Graph

3.3.4 Behaviours

As explained above, behaviours are rules matching conditions on the expectations to actions external to the social reasoner, where actions take the form of adding or removing beliefs to influence (say) a BDI practical reasoner (by restricting the plans in the library that can be currently used).

The set of expectations describes not only current social reasoning, represented in the active expectations, but also possible future belief states and these belief dynamics are encoded in the graph structure. So behaviours may include conditions such as "Φ will always be expected" and "it is possible to expect Φ" in the future.

Considering the expectation graph in Figure 3.4 it is possible to see examples of these two broad categories of behaviour. We will assume that $EXP_A = \{E_1, E_2\}$ – that is, this is the state of the expectation graph the agent is in. Using a notation with CTL modal logic operators (as introduced for $\mathcal{LORA}$) – $\Diamond$ for eventually, $\Box$ for always and **A**, **E** for universal and existential path quantifiers – we can write some behaviour rules:

Condition	$\triangleright$	External Action
$\Phi(E_1)$	$\triangleright$	$add\_belief(A)$
$\mathbf{A}\Box\Phi(E_2)$	$\triangleright$	$del\_belief(B)$
$\mathbf{E}\Diamond(\Phi(E_2)\wedge\neg\Phi(E_1))$	$\triangleright$	$add\_belief(C)$

In all the examples here, the external action is to add a belief to the agent's belief base. The first rules triggers on $\Phi(E_1)$, so with the active set $\{E_1, E_2\}$, if E_1 is current (C holds) it will hold. The second says that $\Phi(E_2)$ is always possible. If we consider the active state, and all those that follow from it, we can see E_2 is always present, so this rule holds. Finally the last one fires when it is possible that eventually $\Phi(E_2)$ and not $\Phi(E_1)$ is active. This is also true, as you can see the state $\{E_2\}$ is accessible from the active set.

Having introduced the main concepts in behaviours with this example, the language used for the components of behaviours can be described in more detail. The rules are of the form:

$$Condition \triangleright Action \tag{3.6}$$

where :

$Condition \in$ *set of well formed CTL formulae*,
with atoms $P \in \mathbf{EXP}$

and :

$Action \in \{add\_belief(A), del\_belief(A)\}$
$A \in$ *set of agent's beliefs*.

The actions $add\_belief(A)$ and $del\_belief(A)$ add and remove a belief from the agent's belief base accordingly.

The set of atoms for the CTL formulae regard expectations only at the meta-level, rather than reasoning over their content. For example, consider a simple case where $\Phi(E_1) = p, \Phi(E_2) = q$ and $p \Rightarrow q$. Should behaviours which have $\Phi(E_2)$ as their condition also apply when the agent holds $\Phi(E_1)$? In this work expectations are considered as atoms and inferences between them not drawn. This reduces the power of the system (for example it may not be possible to check for certain expectation conflicts) but it also greatly simplifies the system. It is in part due to increased simplicity of reasoning that there is some advantage in using ESB, where in fact it does not require the use of logic in actual designs. A possible extension would be to come up with formal infer-

ence rules that are based on tautologies in $\mathcal{LORA}$ and allow reasoning over relational expectation contents, albeit with the usual tractability problems associated with such deductive reasoning in practice – but this is outside the scope of this work.

3.4 Reasoning in ESB

Until this point this chapter has been concerned with the description of the individual components of the ESB framework. The aim of this section is to tie together the previous parts of this chapter and show how all the parts of the ESB theory fit together to allow for agent reasoning. This should not be confused with how an implementation of an ESB reasoner works, although it is a closely related topic. This is covered in the next chapter.

The basic principle behind ESB reasoning is easy to describe. An agent maintains a record of its current expectations, updating them according to its observations. Behaviour conditions can then be checked with reference to these expectations, as restricted by the current strategy. Where a behaviour condition is met, an action can be performed. As ESB is a purely social reasoner, this action would typically take the form of adding or removing a belief from a practical reasoning component's belief base. This is not to say reasoners should not be more tightly coupled, but rather a reflection on the fact that the assumption the practical reasoner bases actions on beliefs is a very general and not overly burdening restriction.

3.4.1 A Reasoning Framework

The central structure in ESB is the expectation graph, or its derived variants. However it is not necessary for a system designer to create this: a lot of the power of ESB comes from the fact that this graph is a general structure and can be generated from a list of expectations. Indeed, more generally, ESB can be thought of as a framework which given a description of some agent reasoning mechanism provides a means to specify and execute it.

To create the complete expectation graph featuring the complete set of states is a fairly simple process. The basic idea is that each possible combination of expectations forms a vertex and where a response is to add some set of expectations and remove some

set, an edge links that vertex to an identical one only with the added expectations and without the removed set. This can be done with the following algorithm:

INPUT: The set of all expectations **EXP**.
OUTPUT: The expectation graph.

Initialise set of vertices: $V \leftarrow \wp(\mathbf{EXP})$
FOR each state $S_a \in V$
  FOR each expectation E_X in S_a
    FOR each response R in $\rho^+(E_X), \rho^-(E_X)$
      add_set = set of expectations added by R
      rem_set = set of expectations removed by R
      $S_b \in V$ is another state in the graph
      Where $S_b = (S_a \cup add\_set) \setminus rem\_set$
        Add a directed edge from S_a to S_b
        Label the edge with $T(E_X)$ for ρ^+
        Label the edge with $\neg\, T(E_X)$ for ρ^-
    END FOR
  END FOR
END FOR

This algorithm represents the relation between states described previously in section 3.3.3. It is a simple algorithm and easy to see how it relates to the previous expectation graph definition. The process creates the set of states as defined previously, using the powerset of all expectations. Then each state is linked to any defined by the expectation responses by considering each expectation in the state in turn. By considering every combination of expectations and every expectation within, it ensures that all states and edges are added.

As this is a naive algorithm that creates the complete expectation graph (with $2^{|\mathbf{EXP}|}$ states) it may be impractical for real-world situations with many expectations. A very basic simplification one can perform to limit its size for practical use is to consider the *accessible* expectation graph, i.e. the graph generated by a search that starts in the agent's initial state and expands the graph until all the edges have been added in the manner described above. Termination occurs when every expectation in every state has been considered and no more vertices are added.

The expectation graph is a fixed structure; an agent need only generate it once, and then simply keep track of the current state. However the strategy graph, defined by the restriction on the response relations, will change depending on current state (and potentially changes in strategy).

3.4.1.1 The Expectation Graph as an FSM

Independent of the actual expectations themselves, a large part of the ESB reasoning *process* can be generalised. To track current state, a finite-state machine can be constructed with one state for each state S in the accessible expectation graph and selecting transitions by applying the $T(E)$ tests for any current expectations (exactly those with $E \in S$ and $C(E)$ holds). As only the expectations current to the state need be checked (since all their dependencies have been "precompiled" into the expectation graph offline), this bounds run-time reasoning further in a useful way.

As well as keeping track of the ESB state, it is necessary to keep track of the expectations to test. This is another area where the ESB approach helps bound agent reasoning, as it is only necessary to test the conditions of the expectations in the current state and infer the appropriate expected beliefs Φ. Combined with only performing tests for the current expectations, an agent can maintain the active state as follows:

INPUT: The active state defined by the set $\mathbf{EXP_A}$.
B, the agent's current beliefs, with no Φ terms.
OUTPUT: The new current state as defined by an updated set $\mathbf{EXP'_A}$

Initialise $\mathbf{EXP'_A}$ to $\mathbf{EXP_A}$
FOR all E_X in active state $\mathbf{EXP_A}$
  IF $C(E_X)$ holds in B **THEN**
    Add $\Phi(E_X)$ to B
    IF $T(E_X)$ holds in B **THEN**
      Update $\mathbf{EXP'_A}$ according to $\rho^+(E_X)$
    ELSE IF $\neg\, T(E_X)$ holds in B **THEN**
      Update $\mathbf{EXP'_A}$ according to $\rho^-(E_X)$
    END IF
  END IF
END FOR

This is a simple algorithm, performing at most a couple of belief checks for each expectation in the current state. This is bound by the (finite) number of total expectations in the worst case (a state with all expectations active) and so despite the simplicity it is not an expensive operation.

The final part of an ESB design concerns the behaviours. These may depend on the current state, or may be conditional on possible future expectations. For the current state it is simple to use the various Φ as the guard on plans at the level of the general practical reasoning, as in the example behaviours given above. The future expectations require a slightly more complex approach. These conditions are likely to take the form of "in all possible cases the agents expects X" or "in some possible case the agent expects X". For the latter, it is simply a case of searching the strategy graph representation for a state matching the condition. For the "all cases" condition, the graph must be searched until all vertices have been checked.

3.4.1.2 An Example of ESB Reasoning

To demonstrate the reasoning process, we revisit the earlier example. The expectation graph and initial state is reproduced for reference in Figure 3.5. The language for the behaviour conditions follows $\mathcal{LORA}$, as introduced in Section 3.3.4. To briefly revisit the syntax, there are two path quantifiers, **E** and **A** mean "there exists a path" and "for all paths". $\Box$ and $\Diamond$ can be read as "always" and "eventually" on a path.

An agent is provided with a specification of all expectations $\mathbf{EXP} = \{1,2,3,4,5\}$ and maintains the sets $EXP_A = \{1,2,5\}$ (active expectations, state $S2$ in the graph) and $EXP_C = \{1,5\}$ (current expectations where C holds, shown in bold).

Other required inputs for the reasoning cycle are:

- The current belief base.
- The strategy in use.
- The set of behaviour rules.
- The practical reasoner ESB is combined with.

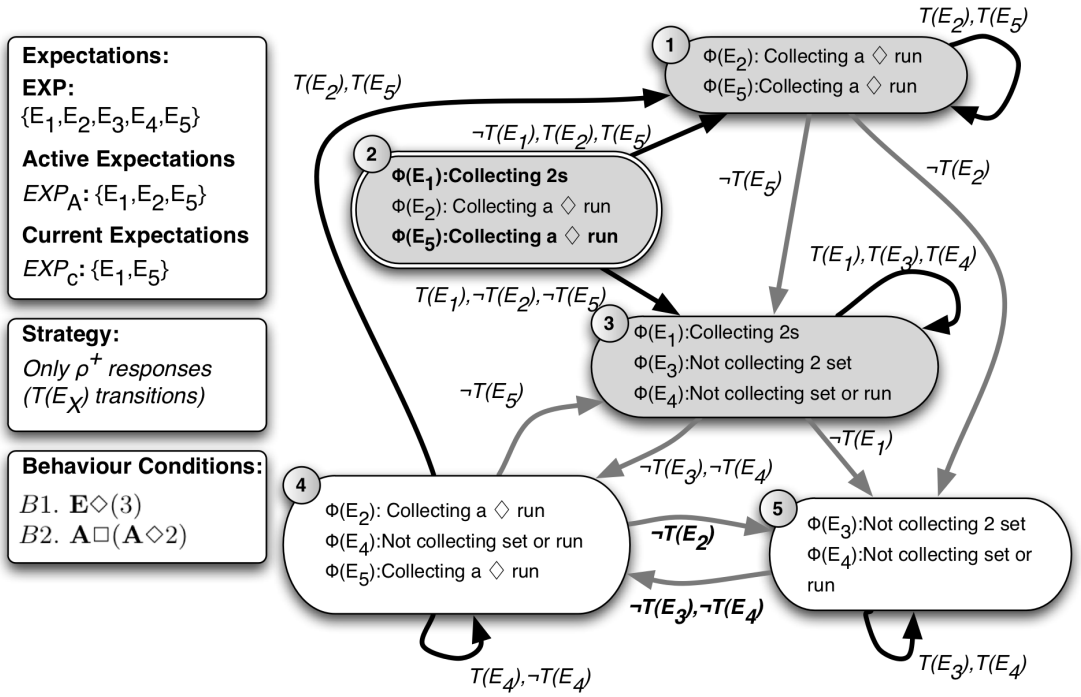

Figure 3.5: The example expectation graph and initial state.

The output from the process is the updated expectation state, belief base and any actions performed by the practical reasoning component.

An ESB agent's reasoning cycle proceeds as follows:

1. Update EXP_C to contain those expectations from EXP_A where C holds in the belief bases. *State S2 is active in the example and the agent holds E_1, E_5 true.*

2. For $E_i \in EXP_C$ add $\Phi(E_i)$ to the belief base. *We add $\Phi(E_1), \Phi(E_5)$.*

3. Apply the strategy to create the restricted strategy graph by considering only a subset of transitions from the current state. *For this example there is an optimistic agent, that considers only ρ^+ responses, those where there is a positive transition in our graph (marked in bold). Applied to the active state S2, this leaves the subgraph of the grey highlighted states as the strategy graph.*

4. For each behaviour, where the behaviour condition holds, add the specified belief (the corresponding *action*) to the belief base. *The example behaviours are as follows; B1's condition says "3 is possible". This is true as state S3 is accessible, and contains E_3 which may become current. B2 says "2 is always possible" and is false, as state S3 is accessible from S2 in the strategy graph and E_2 cannot be current in this state.*

5. Perform practical reasoner plan selection and action execution as normal. This is before expectation responses are applied, in case actions are sensing actions that affect test outcomes.

6. For $E_i \in EXP_C$

   (a) If test $T(E_i)$ applies, update expectations as per the response $\rho^+(E_i)$

   (b) else if $\neg T(E_i)$ applies, update expectations as per the response $\rho^-(E_i)$

   If we assume that $T(E_5)$ holds, then the expectations are updated to place the agent in state S2.

This overview is necessarily high level, the details of the ESB reasoner cycle are part of the implementation. One example of a possible implementation is described in Chapter 4 and was used for evaluation of the ESB framework.

3.5 Considering Nested Expectations

Nested expectations are those that an agent expects another agent to hold. That is, expectations of the form "Under condition C, agent A expects that agent B under condition C_2 will expect Φ ..." and so on. These sorts of expectations are necessary to allow behaviours based on a range of social reasoning situations. For a simple example, consider three agents Alice, Bob and Carl playing the card game "Cheat". Alice plays a card and announces "ace of spades". But Bob holds the ace of spades, so knows Alice is cheating. Under this condition he expects that Carl expects "when Alice says she plays an ace, the ace is played". Bob might then have a behaviour rule that is "play an ace on a big pile of cards, if I expect Carl expects the ace is played" (to make Bob pick up all the cards). This sort of behaviour – where Bob plays according to what he expects Carl to expect – requires nesting. This section describes how the concept of nested expectations can be included in the ESB framework and the implications arising from this.

The first section loosely covers how nesting is incorporated, then goes into more detail on expectations, strategies and behaviours and how adding nesting affects them. Finally algorithms relevant to nesting are discussed.

3.5.1 An Overview of Nesting

At its most basic, a nested expectation is one where the expected Φ is itself an expectation tuple – probably one expected to be held by another agent. Given this, what do the components of the nested expectation mean and how shall they be represented in the ESB framework? Firstly it is helpful to introduce an extra piece of notation to identify a nested expectation. An expectation with name N is labelled E_N. If it is at the base level – not nested – it can be written E^0_N. Where it is at the first level of nesting, it would be E^1_N and so forth. Where the level isn't noted, it can be assumed to be a base expectation. Where N^X is written, this is generally to mean "component N at some level of nesting $X > 0$".

$$\mathbf{Exp}(N^X\ A^X\ C^X\ \Phi^X\ T\ \rho^{+X}\ \rho^{-X})$$

where

N^X is the *name* of the expectation, the main difference here is the nesting level is specified.

A^X is the *agent* holding this expectation. This has more relevance in the nested case compared to the non-nested examples.

C^X is the *condition* under which the expectation holds. This could be a belief, i.e. something the agent holding N^0 can perceive, or an expectation held by the agent.

Φ^X is the expected belief. If this expectation is nested, this is of the form (**Bel** A^0 (**Bel** $B^X\ \Phi^X$)). That is that the agent holding the base expectation that contains this one, believes that the agent expected to hold this expectation believes Φ. The other agent's beliefs are not directly known. Φ may also itself be a complete expectation statement for the nesting cases.

T^X is the *test* which confirms (or rejects) the expectation of Φ

ρ^{+X}/ρ^{-X} are the positive/negative *responses* to the test T^X.

A point to note is that even nested expectations are all held by the agent that holds N^0 really – they are just expected to be held by other agents. In other words, the ESB framework always describes the system of expectations of a single agent, not a system of agents. Nesting allows the representation of its representation of others' reasoning.

The next logical question is what does an expectation graph look like with nesting? This is an important point, as the graphs represent an intuitive way to understand the workings of a set of expectations, and a convenient representation for their use. The simple solution is that there are several complete expectation graphs, one for each level of nesting. Then the current expectation state in level 0 is defined as previously and the nested active set of expectations is that made possible from the current set of expectations. The nested current expectation set of course depends on their individual C^X conditions, which will be either directly observable by the agent or follow from base level-0 expected Φ.

3.5.2 Introducing an Example

To better aid this discussion of nesting and to illustrate the coming sections an example of nesting is presented here. This example is not a real scenario – the set of expectations have been chosen to give a compact yet interesting expectation graph. Firstly we have the expectation tuples, in Table 3.3. For clarity only the relevant items are labelled.

Table 3.3: An Expectation Set with Nesting

N	A	C	Φ	T	ρ^+	ρ^-
E_A	Alice	$C(E_A)$	–	–	$add(E_D)$	$remove(E_A)$
E_B	Alice	$C(E_B)$	E_C^1	$T(E_B)$	–	$remove(E_B)$
E_C^1	Bob	$C(E_C)$	–	$T(E_C)$	–	$add(E_E^1)$
E_D	Alice	$C(E_D)$	E_E^1	–	–	–
E_E^1	Bob	$C(E_E)$	–	–	–	–

The complete level-0 expectation graph and the complete level-1 nested graph are shown in Figure 3.6. Where an expectation's response has no effect, no arc is drawn to ease congestion in the diagram. The next question is, given this set of graphs, what does an accessible expectation graph look like? This is the set of sub-graphs shaded in grey. In the base level, this is just the current state ($S7$ in Figure 3.6) and those reachable from it. In the next level of nesting, this is the sub-graph formed from every state which contains a set of expectations entirely contained within an accessible state in the lower level graph. So in the example this is state $S2^1$, as E_E is expected given E_D, but not state $S3^1$, as E_C is not in any accessible state.

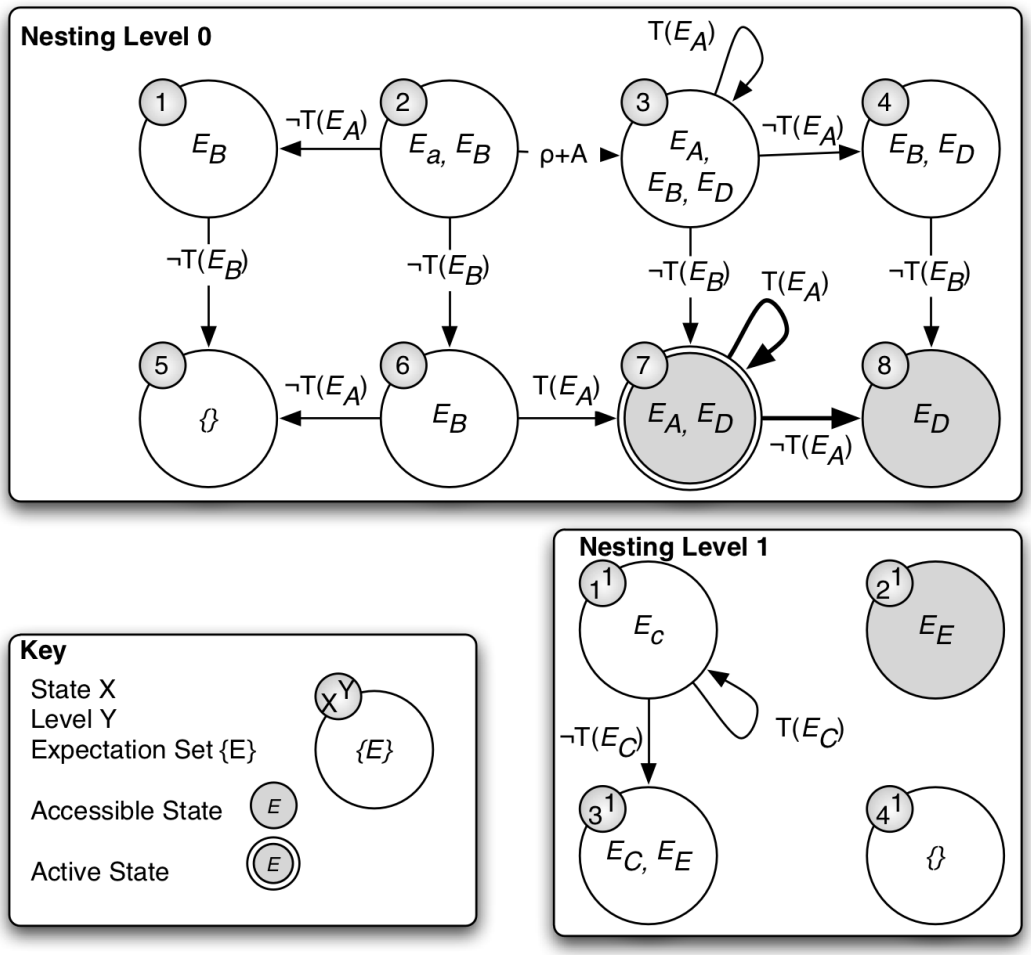

Figure 3.6: Example Nested Complete Expectation Graphs

There are a few potential issues from this treatment of nesting worth considering. What happens if a nested expectation is only present in the response of another nested expectation? E.g. if in this example it was actually the case that $E_E^1 \notin \Phi(E_D)$? A related question would be what about states accessible in a higher nesting level but not present in any lower level? Both these questions actually reduce to the same issue and occur in one of our example states. Consider if the active state was $S1^0$ – what are the accessible sub-graphs? At level-0 it is simply $\{S1, S5\}$. At level-1, it is less obvious. State $S1^1$ is accessible if $C(E_B)$ holds, as E_C^1 is the expectation directly nested by E_B, but what about state $S3^1$? Is this part of the accessible graph? The answer is yes. It is present in $\rho^-(E_C^1)$, which if we recall the formal definition of an expectation means that when $T(E_C)$ is believed false, E_C^1 and E_E^1 are in the active state – or rather we expect the other agent to hold these. But this only occurs if we expect E_B – that is $C(E_B)$ holds. So it is right that the accessible states in the nested graph are included if they're accessible

from any state *cross level accessible* from the lower level.

A state S^X in level X is *cross level accessible* from state S^{X-1} if the set of expectations in it is included in the set of expected Φ of a state at level (X-1).

On a more intuitive level this represents the case where agent A expects agent B to expect one thing on one case and another thing if that is proven wrong. Of course, the alert reader might question could this case not be covered by two different level-0 expectations, or perhaps even without nesting at all? This is an important question and covered later on.

3.5.3 Nesting in the ESB Formalism

What does it mean for an expectation to be nested? Formally a nested expectation is represented exactly the same as in the non nested case, but the Φ term is itself another expectation. Recall the original definition:

$$\mathbf{Exp}(N\,A\,C\,\Phi\,T\,\rho^+\,\rho^-) := \mathbf{A}\Box(\mathit{current}\ \mathcal{W}\ \mathit{update}) \tag{3.7}$$

where

$$\begin{aligned}\mathit{current} =& (\mathbf{Bel}\ A\ C) \Rightarrow (\mathbf{Bel}\ A\ \Phi)\\ \mathit{update} =& \Big((\mathbf{Bel}\ A\ T) \wedge bfA \bigcirc \rho^+\Big) \vee \\ & \Big((\mathbf{Bel}\ A\ \neg T) \wedge \mathbf{A} \bigcirc \rho^-\Big)\end{aligned}$$

and

$$\begin{aligned}\rho^+ &\equiv \mathbf{Exp}(N'\ A'\ C'\ \Phi'\ T'\ \rho'^+\ \rho'^-) \wedge \ldots\\ \rho^- &\equiv \mathbf{Exp}(N''\ A''\ C''\ \Phi''\ T''\ \rho''^+\ \rho''^-) \wedge \ldots\end{aligned}$$

The difference is that the beliefs in the nested expectation are those of a different agent. So the *current* portion looks like:

$$\mathit{current} = (\mathbf{Bel}\ A\ C) \Rightarrow \big(\mathbf{Bel}\ \ A\ \ \big(\mathbf{A}\Box\big(\mathit{current}^1\ \mathcal{W}\ \mathit{update}^1\big)\big)\ \big) \tag{3.8}$$

Where $\mathit{current}^1$ and update^1 are the portions of the nested expectation, defined as usual.

It is worth remembering that we are describing an expectation from the point of view of a single agent, it would not seem appropriate to talk of another agent's beliefs in this

context. However there is no problem in this case, as beliefs are always nested. It is always the case that Φ is of the shape (**Bel** A (**Bel** B Φ)) – A believes that B believes Φ – and this always makes sense from the single agent perspective.

3.5.3.1 Reasoning with Nested Expectations

It is initially worth considering how nesting extends the relevant sets of expectations. First recall the various sets without nesting:

EXP The total set of all possible expectations the agent might hold.

EXP_A The active expectations. This set is one of the possible subsets of all expectations.

EXP_C The current expectation set, those expectations whose condition holds.

So given that these are the sets at level-0 with no nesting, what are the active and current expectations at the next level of nesting?

$\mathbf{EXP^X}$ The total set of all possible nested expectations the agent might hold (or expect) for nesting level X. This is the set of elements where: $\forall E \in \mathbf{EXP^X}$, $E \in \Phi(E') \wedge E' \in \mathbf{EXP^{X-1}}$. That is an expectation in this set must be in Φ of an expectation at the next lower level. *So in the example of the previous section,* $\mathbf{EXP^1} = \{E_C, E_E\}$*, all the expectations of level one.*

EXP_A^X The active expectations at nesting level-X. This is the set of elements where: $\forall E \in EXP_A^X$, $E \in \Phi(E') \wedge E' \in EXP_A^{X-1}$. That is an expectation in this set must be in the Φ of an active expectation at the next lower level. Note that this is also intended to cover the case where it is mentioned in the response of such an expectation (see the previous section for a discussion on this). *In the example, if* $EXP_A^0 = \{E_A, E_D\}$ *then* $EXP_A^0 = \{E_E\}$

EXP_C^X The current expectation set. This breaks a little from the pattern of the previous two sets, as not just the expectation at this level needs to be current, but the enclosing expectation too. This is the set of expectations E where ((**Bel** A C) $\Rightarrow$ ((**Bel** A Φ)) $\wedge$ (**Bel** A C) holds, and also $E \in \Phi(E') \wedge E' \in EXP_C^{X-1}$. *So in the example if* $EXP_C^0 = \{E_A\}$ *then* $EXP_C^1 = \emptyset$ *as there are no nested expectations with a current parent.*

These definitions cover slightly more formally what the active and current expectations are – as described by the example in the previous section. So for our example, where the agent is in state S7 as per Figure 3.6 we have:

Set	Contents
$\mathbf{EXP}$	$\{E_A, E_B, E_D\}$
EXP_A	$\{E_A, E_D\}$
EXP_C	$E \in \mathcal{P}(EXP_A)$
$\mathbf{EXP^1}$	$\{E_C, E_E^1\}$
EXP_A^1	$\{E_E^1\}$
EXP_C^1	$E \in \mathcal{P}(EXP_A^1)$

The current expectation sets cannot be precisely defined without knowing the values for the various expectation conditions. So now the only new problem that nesting leaves us is this: how do we work out these current expectation sets? This is equivalent to working out the exact state and rules that the agent will base its behaviours on, so is of vital importance. Alternatively it can be thought of intuitively as "Given a level-0 state, how do we link it to the corresponding level-1 state?". This is actually quite simple. As per level-0, we just check the conditions of all expectations in the active set. The only difference is that it is perhaps more likely that these conditions will themselves be expected Φ values, though remember there is nothing forbidding this case even without nesting.

Now the nested expectation graph is defined, how do we generate such a structure? For each level the set of vertices is the set of possible active sets of expectations, $\mathcal{P}(\mathbf{EXP^X})$. Edges are then generated using the algorithm described previously in section 3.4.1.

3.5.3.2 Implications for the Expectation Graph

A question that arises from considering examples of this process is "is there more than one nested expectation graph for each level?" Simply there is not. Consider the definition of the expectation graph, it has vertices equivalent to the powerset of all expectations for that level, so every possible expectation state the agent could expect for that level is catered for. However it is likely that there are several disjoint sub-graphs – particularly in the case where the agent may hold different expectations for different third party agents.

Another question is what about more than one layer of nesting? The example here only has two, but all the important sets of expectations, and states, are defined in terms of the adjacent level. Further nesting is defined in the same manner recursively.

Also requiring consideration are expectations where the response is to change (add or remove) an expectation from another level. Firstly this can be the case implicitly. Consider our example above. Moving from state S1 to S5, with the response $\rho^-(E_B)$ removes E_B, and so E_C^1 is no longer in the nested active expectation set. This makes sense intuitively too. If Bob no longer believes the rule "expect Carl to expect Alice is truthful when Alice plays a card", the nested expectation of Carl is clearly part of it. But what about the explicit case? Say $\rho^-(E_B)$ was instead to remove E_C^1, and not E_B? This makes no sense, and breaks the definition of an expectation, as the response is modifying a different level of nesting. This is equivalent to removing E_B and adding another expectation E_B' where the only difference is the change we wish to effect namely in this case remove E_C^1 from Φ. But what about the converse case where E_C^1 removes E_A?This makes no sense, the entire expected expectation is held by another agent, this includes the responses. So the real rule to consider here is: expectations can only modify in their responses other expectations of the same agent. If we consider the formal definition of an expectation this is more obvious. As the responses are not defined as changes, but rather the definition of the new expectations held. If this were not the case, this would require us to write things such as (**Bel** B X) in a definition for agent A (remember ESB only defines the reasoning from one agent's point of view). This cannot be – the closest we could have would be (**Bel** A (**Bel** B X)) – and this would be a different pattern of expectations.

3.5.3.3 Nesting and Strategies

Strategies are a key part of the ESB framework, as they are the mechanism by which an agent's reasoning can be bounded and take advantage of the intuitive graph structure that is a key advantage of defining agent reasoning in terms of expectations. Nesting expectations causes no particular changes to strategies. They work as before, providing some limit on the accessible graph from the current state for the purposes of checking behaviour conditions. This means that any nested expectation graphs will be restricted, as their container expectations are removed according to the strategy in use.

Nested expectations have the potential to make checking behaviour conditions very

expensive, so strategies which reduce the number of nesting levels or expectations checked could be very useful. To do this, strategies could exploit the knowledge of the lower level container expectations and how they change, to simplify a nested level.

For example, it is reasonable for an agent's strategy to be that it assumes the other agents follow some strategy. So for example if Bob believed Carl was playing according to minimax strategy, the appropriate strategy to represent such play could be represented by applying restrictions to the test result transitions that would indicate play inconsistent with this strategy at the lower level of nesting.

3.5.3.4 Nesting and Behaviours

As mentioned in the introduction, the main reason to include nesting in ESB is to allow for behaviour that is conditioned on how an agent expects another agent is reasoning. Intuitively this is required for situations such as those where common knowledge is an important concept. So it is necessary to explore which behaviour conditions can now be captured that were not possible before. It is principally behaviour that is not only dependant on what we expect, but dependant on how others are reasoning and will reason in the future. This can be made clearer if we consider each additional type of behaviour condition in turn that nesting gives us. Table 3.4 below exemplifies the different classes of condition that result from the general framework, for one level of nesting. Obviously the combinations increase as the numbers of layers increase, but the pattern is evident here. For clarity, in this table $P\mathcal{X}\Phi$ is used for "P can expect Φ", **A** to denote "All Accessible States"(AAS) and **E** to denote "Exists an Accessible State" (EAS). The two agents used are P and Q, and the expected belief is Φ.

It is fairly obvious from this table that there is a pattern. There are fundamentally three sorts of behaviour condition and so with 2 layers we get 9 combinations. Each of the three conditions is checked in the same way, regardless of whether it is nested or not. The only difference is that if it is nested once, then it is checked on the level one graph and so on. So for each of the three patterns, how do we check them in the nested case? It is similar to the non-nested case, we just use the various expectation set definitions for the appropriate level. So we have:

$Q\mathcal{X}\Phi$ – Q expects $\Phi(E)$ when E is in the current set of expectations (so in graph terms, E is in the current state, and $C(E)$ holds).

Table 3.4: Example behaviour conditions made possible with nesting.

Behaviour Condition	In English
$P\mathcal{X}(Q\mathcal{X}\Phi)$	P expects Q expects Φ
$P\mathcal{X}(Q\mathbf{A}\mathcal{X}\Phi)$	P expects that Q expects Φ AAS
$P\mathcal{X}(Q\mathbf{E}\mathcal{X}\Phi)$	P expects that Q expects Φ EAS
$P\mathbf{A}\mathcal{X}(Q\mathcal{X}\Phi)$	P expects AAS that Q expects Φ
$P\mathbf{A}\mathcal{X}(Q\mathbf{A}\mathcal{X}\Phi)$	P expects AAS that Q expects Φ AAS
$P\mathbf{A}\mathcal{X}(Q\mathbf{E}\mathcal{X}\Phi)$	P expects AAS that Q expects Φ EAS
$P\mathbf{E}\mathcal{X}(Q\mathcal{X}\Phi)$	P expects EAS that Q expects Φ
$P\mathbf{E}\mathcal{X}(Q\mathbf{A}\mathcal{X}\Phi)$	P expects EAS that Q expects Φ AAS
$P\mathbf{E}\mathcal{X}(Q\mathbf{E}\mathcal{X}\Phi)$	P expects EAS that Q expects Φ EAS

$Q\mathbf{A}\mathcal{X}\Phi$ – Q can expect $\Phi(E)$ in all accessible states. This is the case when E is in all the candidate states reachable from the lower level candidate state (according to the strategy). The candidate state at the base level is the current state, defined by the active set EXP_A. The relation between states is defined by the expectations' responses.

$Q\mathbf{E}\mathcal{X}\Phi$ – Q can expect $\Phi(E)$ in some accessible state. This is the case when E is in one of the candidate states reachable from the lower level candidate state (according to the strategy).

The difference to note here is the use of a "candidate state". In a regular non-nested case it suffices to consider only the current state as the candidate as we know that's the state we're in. But when we have a condition like $P\mathbf{A}\mathcal{X}(Q\mathcal{X}\Phi)$, we need to check not just for $\Phi(E)$ in the current state (in the level-1 graph) but in the set of all possible nested active states that are possible from the states in the current accessible graph at level-0. It is only when this is realised that one realises the true enormity of the problem of checking complex nested expectations. In this case we are basing our behaviour on what we expect another person could expect, in all possible future reasoning states we might find ourselves in. Clearly this is incredibly complex reasoning!

Some behaviour condition examples will clarify this. The below examples all consider the above example graphs and expectations, assuming that as in the diagram state $S7^0$ is the active state and for the sake of the example the current set shall be $EXP_C = E_D$.

$Alice\ \mathbf{A}\mathcal{X}(\ Bob\ \mathcal{X}\ \Phi(E^1_E))$ **holds** – In all possible future reasoning states, Alice could

expect that Bob believes $\Phi(E_E)$. This is true, as in all level-0 states E_D is present and this is the necessary condition for E_E^1 to be held.

Alice $\mathbf{E}\mathcal{X}($ *Bob* $\mathbf{A}\mathcal{X}\ \Phi(E_C^1))$ **does not hold** – In some future reasoning state, Alice expects that Bob may always be able to expect E_C in all his future reasoning states. This is false, as in no reachable state in level-0 is there a corresponding level-1 state with E_C. For an example state where this is in fact true, consider if the active state was $S2^0$. In this case, states $S2$ and $S1$ both contain the nested case that E_C is always possible.

It is hard to define the problem of behaviour conditions precisely without a specific language of behaviour conditions, which are dependant on the language used for implementation. But the approach is to create the minimally prescriptive ESB framework, to allow representing the most general conditions. In any case, this is hard to do without concrete cases to consider and allow for. However, if necessary for a specific implementation or example it would be possible to determine what this language would be – as the other parts of the framework would be defined specifically enough to allow this. One such concrete language is provided by the ESB-RS implementation described in Chapter 4, but this is not an integral part of the theoretical framework.

3.5.4 Algorithmic Considerations

As explained above, most of the algorithms that would be required to design an agent reasoner with nested ESB principles are the same as for the non-nested case. The main new challenge brought by nesting is calculating the candidate states described above and the current state in nested graphs (though this is actually a subset of the candidate state problem). To recap, the candidate states are those possible active sets (states) at the next level of nesting. So in our example, if we consider the candidate states from state $S4^0$ they are $\{S1^1, S2^1, S3^1, S4^1\}$ – the complete nested expectation graph. So to check the condition *Alice* $\mathcal{X}($ *Bob* $\mathbf{A}\ \mathcal{X}\ \Phi(E_C^1))$ for state $S4^0$, we would need to check for $\Phi(E_C^1)$ being possible in each of these states. A naive algorithm to calculate the candidate states, given a state in the lower level graph is as follows (where states are defined by sets of expectations):

INPUT:

$\{E_A, E_B \ldots E_n\} \in EXP_A^X$, active expectations, current graph vertex.

$EXP_{candidates} = \emptyset$, the candidate active sets (states in level $X+1$)

$\mathbf{EXP^{X+1}}$, Set of expectations in level $X+1$

OUTPUT:

$EXP_{candidates}$, candidate states.

FOR every possible set $EXP_{possible} \in \wp$ $\mathbf{EXP^{X+1}}$
    IF $EXP_{possible} \subseteq EXP_A^X\Phi$, where $EXP_A^X\Phi = \{\Phi(E_A), \Phi(E_B) \ldots \Phi(E_n)\}$
        THEN $EXP_{candidates} = EXP_{candidates} \cup EXP_{possible}$
    END IF
END FOR

What this means in natural language is:

FOR every possible subset of expectations in the nested level
    IF this subset is in the active state's expected Φs,
        THEN add it to the candidate set
END FOR

This is process is presented as a simple algorithm so it can easily be seen how it work. It is however an expensive algorithm – as every possible subset of the total set of nested expectations must be inspected. A trivial improvement would be to work the algorithm the other way round, and generate all possible "successor" states from the container Φs at level X.

3.5.5 Discussion of Nesting

So to conclude, the changes made to include nesting are minimal. Most of the algorithms remain unchanged, and the components of ESB are defined similarly – there are just more (separate) graphs. The only new trick is how the relations between these graphs are handled and this reveals one of the weaknesses of the graph intuition for ESB. The expectations are dynamic and the graph only captures some of this change. This is most evident when you consider the differences between the current set and the active set and how they both define the accessible parts of nested graphs slightly differently, depending on what conditions are being checked (AAS and EAS, or cur-

rent expectations). However the graphs and graphs of graphs approach is useful and the information and structure it reveals will help to create useful algorithms that can be applied.

3.6 Summary

This chapter has introduced the ESB framework as a proposed solution to the problem of specifying agent social reasoning in a general fashion. Starting with an example chosen to best illustrate the features of the framework, the chapter then moved on to the formal definition of each part of ESB. This covered the properties of an ESB specification of reasoning, so the next topic covered was the algorithms required for reasoning based on a specification. Finally the extension to nested expectations was explained and discussed.

At this stage in the thesis it is not possible to evaluate the ESB design and formalism properly, this will come later in Chapter 5 after the implementation and case studies have been introduced. It is however possible to have some discussion of the design and why it takes the form it does. The most striking property of ESB is its generality, very little is defined precisely. This was intentional and there are two reasons for this. Firstly, the only purpose of the formalism was to allow for considerations of exactly what an expectation is when designing the framework. This means that a "strict" formal specification was not needed. As discussed earlier this would have been a very different direction for the work, the eventual goal here was implementation to show that the idea of a separate social reasoner was practical and possible to implement. These were the reasons for the lack of specificity in the formalism. There is a second sort of generality evident here, the ESB framework as defined has very few constraints on the reasoning of the agent. This is also intentional and necessary. The purpose of ESB is to provide a framework to specify reasoning, not limit the style of reasoning in any way. If it did, then it would restrict the sorts of reasoning schemes that can be implemented. Additionally, whilst the purpose of this specification of the ESB framework is to create an ESB reasoning engine, it is also desirable that the specification only guides the properties of the design and does not over constrain the implementation. The inspiration for this approach comes from the success of BDI theory for agent implementation and design. As discussed in the background chapter, a portion of the success of BDI is clearly attributable to the way in which it guides the agent designer

without being overly prescriptive in the details of how the agent is implemented. This is a desirable property and one worthy of emulation. Over-generality is always a risk, but there is no correct choice here. The choice made was to err on the side of increased abstraction.

An avenue of further research that there was not time to pursue during development of this thesis was that of establishing if it is possible to rewrite a set of expectations with nesting into a set without – but allowing equivalent behaviour conditions to be checked. However, even if it is possible to show that nesting is not needed, and sets of expectations can be rewritten without the nesting, it is worthwhile to include discussion of nesting. The purpose of showing that rewriting is possible in the general case would only be to show that properties more easily provable for a non-nested graph are true of nested sets also. The nesting does give the agent designer something valuable – and that's the ability to write down an agent's reasoning scheme in a simple, intuitive manner – for the same reasons as when there is no nesting.

The description of a scheme for generic social reasoning at a theoretical level was the second major task of this thesis after the background review and thus it has been presented here to set the scene for the implementation, case studies and evaluation of ESB that will follow. The specification of an ESB agent has been described and the processes by which it updates itself. The exact operation of an instance of an ESB reasoner is the next logical step and will be described in the next chapter, where one possible implementation of ESB is introduced.

Chapter 4

Implementation of an ESB System

This chapter contains a description of the design and implementation of ESB-RS (ESB Reasoning System). The presentation of a theory for general social reasoning in the previous chapter is not in itself a sufficient argument that separate general social reasoning is practical and beneficial. Creation of an implementation shows that it is indeed practical. The design of ESB-RS then provides a concrete setting to discuss the pros and cons of ESB, the problems arising from creating ESB agents and showcase the benefits. Case studies, presented in Chapter 5, are used to evaluate the suitability for ESB to capture different types of reasoning and a working ESB reasoner was required to create these. The combination of these evaluations of the implementation and the implementation itself show that the ESB framework is general and the required reasoning processes can be carried out algorithmically.

Computationally, to reason with an ESB specification requires generating a model of states that satisfy the agent specification, reducing this model to meet the strategy and then using it to check behaviour conditions. ESB-RS is one possible implementation of this concept, based on using a model checker to build a representation of the expectation graph and allow a natural way to check behaviour conditions. To create a complete agent reasoning system ESB is coupled with the Jason [Bordini et al., 2007] BDI reasoner, which is easily extended in Java. This chapter will start with the higher level design of the ESB-RS system, before going on to a description of the more interesting details of the implementation.

Much of this chapter is based on the work presented previously in Wallace and Rovatsos [2010].

4.1 Design of an ESB Reasoner

The previous chapter has defined ESB and what an ESB specification looks like. This section presents one possible design of an ESB social reasoning engine. As ESB is a generic framework for *social* reasoning, it must be combined with a practical reasoning component. The main components of a complete ESB agent are:

- A BDI reasoning engine, in this case Jason [Bordini et al., 2007].
- A set of plans for practical reasoning, here they are specified in AgentSpeak(L) [Rao, 1996].
- An ESB engine, to maintain the expectations and interface with Jason.
- An ESB specification in terms of its expectations, strategy and behaviours.

Of these only the last has been presented in detail in the previous chapter. The contribution here is a description of the algorithms required for the ESB engine and coupling of this engine with the BDI interpreter. The interactions between these components are illustrated in Figure 4.1. This overview shows the similarity with the general overview of the framework shown in Figure 3.1 and is the same framework with components made concrete for implementation. The principal interaction between the social and practical reasoning components is via the belief revision function (BRF). Expectations are updated based on the agent's perceptions and beliefs. The control of the agent's social interactions is also in terms of beliefs, as these can act as guards on plans to carry out actions. At a high level, the execution of ESB-RS can be described as follows:

1. The expectation graph is generated from the specifications. This is only done once, as after this initial step the agent can simply track its current state in the graph.
2. In each reasoning cycle, the beliefs from the BDI component, this expectation graph and the strategy specification are used to create the reduced strategy graph.
3. The strategy graph is then used by the behaviour condition checker to select the applicable behaviour actions, which are used to update the agent's beliefs accordingly.

The next step is to consider each part of the ESB system in turn and how it can be captured in a reasoner design.

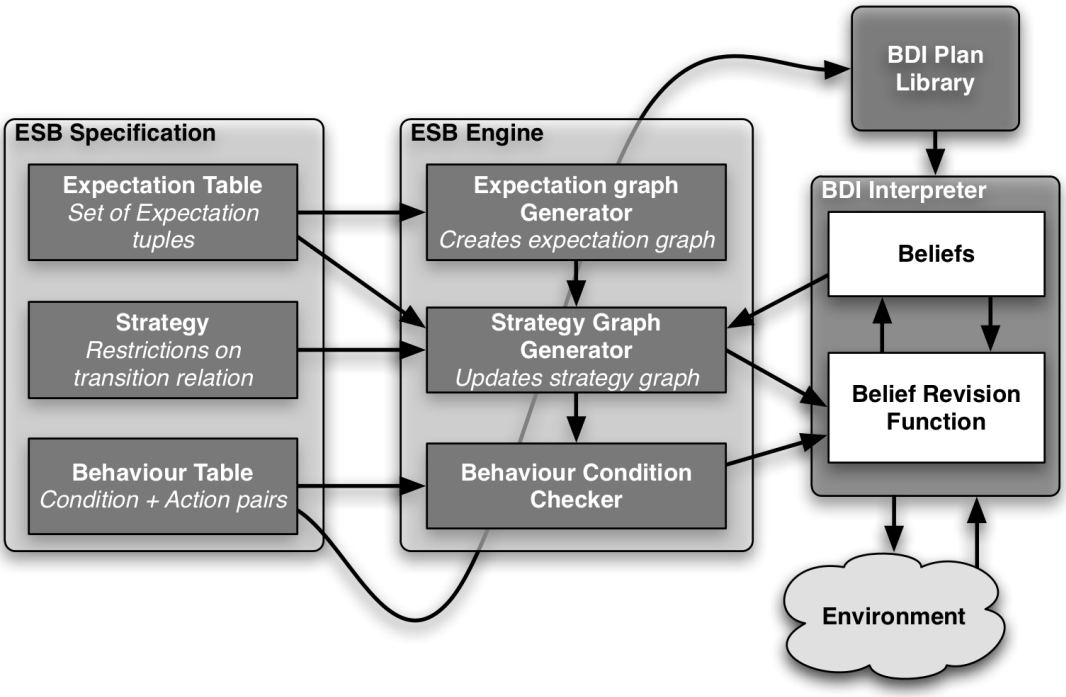

Figure 4.1: Conceptual Overview of an ESB-RS Agent

4.1.1 Expectations

The agent's current state is captured by three sets of expectations: current, active and inactive, updated according to the algorithm described previously in section 3.4.1.2.

In ESB-RS the expectations are transformed into a finite-state machine (FSM) representation at a lower level of abstraction than the expectation graph (as each state in the expectation graph represents several possible combinations of current expectations). Behaviour conditions are however defined in terms of the expectation graph. Accordingly it is necessary to transform a declarative expectation specification into a form suitable for efficient behaviour checking. The graph may be thought of as a FSM, where it is necessary to check if certain properties hold (the behaviour conditions) given an initial state (defined by EXP_C). This intuition naturally suggests the approach of model checkers, which aim to solve a similar problem. This requires algorithms to create the state descriptions and transition relation from the sets of expectations.

In the state description for the model checker, there is no need to explicitly consider all the combinations of expectations that form states in the expectation graph. This is one of the advantages brought by the model checker, as it implicitly considers these combinations. The FSM is defined at the level of individual expectations, which are

specified more easily. Testing a condition on a particular expectation graph state, or setting the current state, is done by specifying the state of all individual expectations. This finer grained approach is possible as each state in an expectation graph can be considered not as one state, but as a high level abstraction of the set of states where only a subset of expectations are relevant. This transformation allows the use of the NuSMV [Cimatti et al., 1999] model checker and ultimately makes it possible to easily implement the ESB theory.

Creating the FSM requires only the components of each expectation that describe the dynamics of the reasoning. The possible states of each component are as follows:

Condition $\in \{True, False, DC\}$ – Each expectation can be considered to be $True$ or $False$ (holds or does not), if it is in the active set EXP_A, or "don't care" (DC) if it is not. Expectations in EXP_C are of course $True$.

$\Phi \in \{True, False\}$ – This is as per **condition**, or $False$ where condition is DC.

Test $\in \{Tp, Tm, NA\}$ – The test is either positive (Tp) or negative (Tm) or not-applicable (NA). It is not-applicable whenever the expectation is DC.

Responses ρ^+, ρ^- – The add- and remove-sets of expectations for each response are used to define the transition relation of the FSM.

The set of expectation graph states translate to the FSM as the set of all unique combinations of each expectation's condition (though of course many will be unreachable). From this it follows that it is only necessary to specify how each individual expectation changes to define the FSM transition relation and the complete graph is captured. Each expectation is specified in terms of its variables, and expressions for the next state in terms of the current state variables. This is done as shown in Figure 4.2.

The algorithm in Figure 4.2 does not represent actual implemented code, but captures how the specification of the FSM for the NuSMV model checker is procedurally specified. The actual ESB FSM specification code can be found in Appendix A. The problem of actually creating the FSM for the purposes of model checking from this description of its transition relation is handled by the model checker. This is a key benefit of this approach. As it is a simple operation to generate this specification and the complexity of the problem of building the FSM is handled by the model checker.

Each expectation is defined in terms of its component variables and how they change from state to state. The algorithm is tricky to understand, but easily described in a more

INPUT: The set of all expectations **EXP**.
OUTPUT: An FSM, specified in terms of each expectation's dynamics.

Each expectation E ∈ **EXP** has components:
E.Condition ∈ {True, False,DC}
E.Test ∈ {Tp,Tm,NA}
The sets of expectations added and removed by responses:
E.Tp.addSet,E.Tp.removeSet,E.Tm.addSet,E.Tm.removeSet

Next state for each E calculated as follows:
FOR every other expectation O ∈ **EXP**
IF (O.Test = Tp) & (E in O.Tp.addSet) **THEN**
E.Condition ∈ {True,False}, **RETURN**
ELSE IF (O.Test = Tm) & (E in O.Tm.addSet) **THEN**
E.Condition ∈ {True,False}, **RETURN**
ELSE IF (O.Test = Tp) & (E in O.Tp.removeSet) **THEN**
E.Condition ∈ {DC}, **RETURN**
ELSE IF (O.Test = Tm) & (E in O.Tm.removeSet) **THEN**
E.Condition ∈ {DC}, **RETURN**
END IF
END FOR
IF E.Condition ∈ {True,False} **THEN**
E.Condition ∈ {True,False}, **RETURN**
ELSE IF E.Condition = DC **THEN**
E.Condition ∈ {DC}, **RETURN**
END IF

When the condition is assigned and returns, assign the test:
IF E.Condition ∈ {DC} **THEN**
E.Test ∈ {NA}
ELSE
E.Test ∈ {Tp,Tm,NA}
END IF

Figure 4.2: Algorithm Defining a FSM from an Expectation Set

natural manner. The for-loop says that if this expectation is in an add-set of another expectation where the relevant test applies, then it will become active (`True` or `False` – it may or may not be current). Similarly if it appears in a relevant remove-set then it will be set to `DC`. If neither case applies, each expectation's condition may at the next step change between `False` or `True`, however if it is `DC` then it stays as such. The test for an expectation is not applicable (NA) if the expectation is inactive, otherwise it may be the positive case (Tp), negative (Tm) , or neither (NA).

4.1.2 Strategies

Strategies are defined as restrictions on the expectation graph, to reduce the search space when checking behaviour conditions thus bounding the agent's social reasoning. This is a function from one graph to another, where either the edges or vertices are reduced in some way. So in this implementation it is the number of accessible states in the FSM that must be reduced to simplify checking. Strategies are so called, as they influence the overall style of the agent's social reasoning rather than being concerned with the details. A simple example would be an optimistic agent, which considers only the positive response transitions, effectively assuming its assumptions about other agent's mental states are always correct. Alternatively it is possible for strategies to have a greater influence on an agent's behaviour. For example, the expectations could be restricted to those consistent with a particular opponent model.

Strategies are implemented as a set of constraints on the transition relation, which can be conditioned on either the edges (the tests that trigger transitions) or the states (the expectations that define these). Each constraint is a boolean expression which must be satisfied in all states in the strategy graph and so the expectation graph is reduced to those states satisfying these constraints.

$$\begin{aligned} \textit{Constraint } \varphi \quad ::= \quad & \top \mid \bot \mid c \mid t \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \\ \textit{where} \quad & c \in \{C(E_1), C(E_2) \ldots C(E_n)\} \\ \textit{where} \quad & t \in \{T(E_1), T(E_2) \ldots T(E_n)\} \end{aligned} \tag{4.1}$$

This means the transitions from a given state are constrained to those where the next state meets all the strategy constraints. The atoms in the constraints are the conditions and tests of each expectation. These allow restrictions to the states and transitions respectively. These can be combined using the standard propositional logical operators.

For example, the graph could be restricted to consider only those states reachable assuming tests, if they apply, always hold. The constraint to achieve this would be to limit the next state as follows:

$$T(E_1) \wedge T(E_2) \wedge \ldots T(E_n) \tag{4.2}$$

This says that for any state, in the next state the tests hold for each expectation (there are by definition a finite number), if it applies at all.

4.1.3 Behaviours

Behaviours are defined in ESB as condition-action pairs. Actions in behaviours simply result in belief change in the BDI reasoner's belief base and thus control and bound the agent through guards on its plans. This has advantages from an implementation simplicity point of view and also conceptually, separating the abstract social reasoning from the more concrete practical. There is a downside to this intertwining however, the concept of a "behaviour" is split between its specification and the plans involving it. The blurring of this dichotomy is not without precedent, however. In most PRS [Georgeff and Lansky, 1987] derived BDI reasoners, of which Jason is one of many, an agent's desires are not represented explicitly, but are encoded in the plans and intentions of the agent.

The conditions may capture a combination of current and future expectations. It may be desirable, for example, to only pick up one end of a table if I expect another to lift the other end. Or I might only discard a card in a game if I do not expect the other agent to be collecting it, and cannot expect them to do so in the future.

As written previously constructing a graph from a set of expectations and checking conditions based on the current state and possible (or impossible) future states naturally suggests an approach based on model checking. This presents an attractive advantage – it is only necessary to describe the states each expectation may have and the relatively simple relations between them – the model checker will then allow us to form relatively complex queries about expectation graph concepts.

NuSMV is chosen as it allows the checking of CTL logic formulae, allowing us to express possibilities and necessities. A simple example is the condition "I might hold expectation 2 in the future" which is captured as $\mathbf{E}\Diamond(\Phi(E_2))$ (there exists a path where eventually $\Phi(E_2)$).

4.2 Implementation of an ESB Reasoner

Following on from the overview of the ESB-RS design, this section provides more implementation details and commentary on the design process. The aim is to give a more detailed impression of how ESB-RS works and therefore present a better picture of ESB's functionality. It is important to keep clear the distinction between ESB as a theory and ESB-RS developed for this thesis to evaluate the ESB theory. ESB only specifies the components of expectations, how strategies affect them conceptually and how behaviours translate from expectations into a practical reasoner. ESB-RS must go further than this to define a complete agent. So ESB-RS describes an ESB + BDI agent and how that interaction is implemented. Other practical reasoning schemes could have been used and the choice of BDI and specifically Jason is explained next.

4.2.1 The Practical Reasoner – Jason and BDI

The choice of BDI for the practical reasoning component was an easy one. It is arguably the most generally recognisable and studied method of practical reasoning in MAS. Due to the maturity of the theory, and its prominence in MAS research for many years there are a variety of existing implementations to choose from. The BDI interpreter chosen is Jason [Bordini and Hübner, 2009] as it has many desirable properties:

Well-documented In addition to many provided examples, there is a book [Bordini et al., 2007] and several papers describing its implementation and operation.

High Level The plans are written in a version of AgentSpeak(L) [Rao, 1996], which is a high level language specifically designed for agent programming. Therefore programs can mention beliefs and intentions as first-class concepts.

Modular Java implementation The interpreter has been designed to be very flexible and easy to extend, as it is split up into several components. This is an advantage, as it means integrating an ESB reasoner is just a case of implementing a new belief revision function.

Under active development The first release of Jason was over 5 years ago and it is still in active development with bugs being fixed and new features added. This means it is a reasonably high quality implementation and reasonably well tested.

4.2.2 The ESB Specification

There are two main tasks to creating an ESB "engine". There is the implementation of the algorithms necessary to maintain the expectation graph and check the behaviours and there is the specification of the expectations, strategies and behaviours themselves. This section concerns itself with the latter. An important concession is that what is realistic and simple enough for an implementation may be more restrictive than the formalism. Where assumptions have been made for simplification, this will be noted.

There are three main components to the ESB specification itself, which will each be considered in turn here. Technical details can be found in Appendix A, which contains the XML schema for each component.

4.2.2.1 Expectations

These serve as input to both the expectation graph generator and the strategy graph generator. The expectations are needed to update the strategy graph as it must also handle the updates to the system and the strategy graph represents the current state of the system. Although the implementation uses a model-checker based approach, the specification of the expectations are independent of this.

The components of an expectation are specified in the following terms:

N – One of the set of names of expectations, a string.

A – A string representing an agent's name. Should be one of the set of names defined in the Jason MAS specification being used.

C – A propositional statement about beliefs. This is an AgentSpeak(L) term that will be evaluated on the agent's belief base by Jason.

Φ – A belief, which will be added to the agent's belief base when C holds and removed when C does not.

T+ – Defined as per **C**. The formalism for ESB defines the test in terms of two parts, T and $\neg T$. For implementation this is taken further, with the two tests defined separately. There is a positive $T+$ case and negative $T-$ case. This allows the negative test cast to not be strictly $\neg T$, which is useful in cases such as specifying a timeout. For example, for a timeout it may be that some condition T is met

(a parcel is delivered), or that some other negative condition is met (that it is a certain time).

T− – Defined as **T**+.

ρ^+/ρ^- – Each of the positive and negative set should contain an *add set* and *remove set*, which are the expectations that are added and or removed when the response applies. These sets are sets of expectations labelled with the same names as used in **N**.

An extension that greatly reduces the verbosity of expectation specifications is the support of variables in the various terms. This allows for writing general expectations, covering a wide set of circumstances. It is not a full first-order extension, it is limited to allow for a relatively simple implementation, but still proves effective in greatly speeding the task of writing expectations. It is implemented by grounding all possible variables when the condition C is checked and applying the resulting matches to the same variables when Φ is added to the belief base, or the tests carried out. As variables effectively represent short-hand for sets of expectations, there may be multiple instantiations. If there is more than one valid variable binding for C, then Φ is added to the belief base once for each binding. The tests are also considered for each binding, exactly as if there were instead a set of ground expectations.

To clarify, consider the following simple blocks-world example. The agent has an active expectation where:

$$\begin{aligned} C &= block(X) \\ \Phi &= move(X) \end{aligned} \tag{4.3}$$

Table 4.1 then shows what Φ are added to the agent's belief base as its beliefs change. Note that for practical purposes, beliefs added from expectations or actions by ESB are annotated by their source. Lowercase indicates a ground atom and uppercase a variable, as per Jason syntax.

In the case where an expectation is not completely ground, the expected Φ is added with variables, so any test which will allow successful unification against the unground Φ will succeed. So in time step 4, a plan with a guard `move(c)` would be applicable, as Y would bind to `c`.

Table 4.1: Expectation Condition Unification

Time Step	Belief Base	Added Φ
1	block(a)	move(a)$_{\Phi\{X=a\}}$
2	block(a) block(b)	move(a)$_{\Phi\{X=a\}}$ move(b)$_{\Phi\{X=b\}}$
3	block(b)	move(b)$_{\Phi\{X=b\}}$
4	block(Y)	move(Y)$_{\Phi\{X=Y\}}$

4.2.2.2 Strategies

To describe the implementation, and thus limitations, of strategies in ESB-RS more detail is required. The specification of a strategy in detail is shown in the XML schema and FSM templates used, both are presented in Appendix A and a higher level overview of these is presented here.

The specification of the complete FSM is split into one main module and then a sub-module per expectation. The main module controls the update for each sub-module and handles carrying out the tests and thus effectively controls the change in which expectations are active. The specification of strategies allows the user to add transition constraints at both the main module and individual expectation level. The main level allows constraints on the test results (and thus responses) that will be considered and the expectation level constraints can be applied to the update of the expectation conditions. This maps to the test and condition atoms described where the strategy implementation is introduced earlier in section 4.1.2.

In the following examples a syntax very close to the implementation is used. The constraints are written to describe conditions that must hold from one state to the next for the variables representing expectations. The keyword `next` describes conditions that must hold in the next state, possibly based on conditions in the current state.

An example main-module constraint to remove negative response transitions from the strategy graph, equivalent to the example earlier, would be:

```
next(Test != Tm)
```

This says that in all successor states `Test` cannot be the negative case. Note that for ease of implementation it is possible to specify that all tests must follow a certain form, or to refer to specific expectations. An expectation module constraint to ignore

a particular expectation (i.e. set the state to "don't care") is as follows:

```
(name = Exp3) -> next(ExpC = DC)
```

The example here says that if the value of `name` for an expectation is `Exp3` then in successor states the condition must be "don't care" – the expectation is not active. This is only one possible way the transition relation could be specified by the strategies and was chosen for ease of implementation and to be minimally constraining. If more complex strategies are required, the modifications can be made in the FSM template file, without requiring that the ESB-RS reasoner be modified.

4.2.2.3 Behaviours

Behaviours are an important part of ESB, as they form the principal link between expectations and the practical reasoner, allowing an agent designer to condition behaviour on complex conditions. As such, the language of behaviour-conditions imposes real limits on the power of an ESB agent. The more complex the conditions that can be checked, the more complex reasoning can be captured. The motivation was then to provide the most general language for conditions within the constraints of the model checker. A behaviour is specified in terms of a condition and an action. The action is simply a belief to add to the agent's belief base. This can then be used however the designer sees fit by practical reasoner plans. For example, it could act as the guard on a plan to request a joint action (this is exactly the case in the example presented in the next chapter).

For implementation purposes the conditions are split in two. The CTL component, which is a NuSMV syntax CTL condition to check and a Jason condition, which is a Jason-syntax expression evaluated on the agent's belief base. This allows for behaviours to account for future reasoning (through the CTL condition) and the current state of the agent's belief base. This addition of a condition on the belief base to the condition on the expectations described in the theory serves principally to ease implementation of agent systems and make them more efficient. Instead of adding an action belief then checking it in the practical reasoner before committing to interaction, the condition can include this check within it and further more this (cheap) check can be done before a (more expensive) model checking operation is committed to. This is not a functional change from how the ESB framework is described, but it allows for more efficient implementation.

There are limitations however, imposed by the model checker. The model checker is not first-order and so variables cannot be used. Effectively this forbids behaviours conditioned on the content of future expectations. This is easier to explain by example. An agent has an expectation E_P which says when to expect another agent is lying. The intended behaviour is that if it expects the other agent may lie in the future it will refuse to trade with them. With this limitation, the designer must write a behaviour that explicitly checks for E_P (and any other expectations to do with lies). Ideally, the condition would simply be on "lying" directly and any expectation that matched would be considered. The implications of this are discussed later in Chapter 5.

4.2.3 Integration with the BDI Reasoner

Jason is designed to be easily extended and so made an excellent choice for the ESB-RS implementation. This was carried out by extending the default agent implementation to carry out an ESB update phase at the start of each reasoning cycle. This is run after the agent's Belief Update Function (BUF) so that the percepts are up to date, and before any explicit belief revision takes place. The reasoning loop used to update the expectations and apply the changes to the agent's belief base functions as follows:

1. Update percepts using the standard BUF.
2. Check all EXP_A, and where C holds add to EXP_C
3. Create the FSM specification using EXP_A, EXP_C to define current state, and include the strategy.
4. Run the model checker to verify the CTL condition for each behaviour, and check the Jason-condition against the belief base.
5. Add the relevant action-beliefs to the belief base from those behaviours that are verified true.
6. Perform the tests for all $E \in EXP_C$.
7. From all the tests that apply, choose one randomly and apply the response.
8. Re-check all EXP_A, and where C holds add to EXP_C.

The last step is also performed for each belief update. This is important, as it ensures that within a reasoning loop an agent's expectations remain consistent with each other.

If this were not the case, updates in the expectations would take two reasoning cycles to update. As it is, changes to expectations (especially those dependant on other expectations) update the same cycle, making debugging agents far easier.

Step 3 does not have to create the entire FSM each step. When the agent is initialised the expectations are parsed into the FSM format. It is necessary in each step to write out the FSM specification prepended with the current state and any strategy. Doing this per-step would allow for more complex strategies that change with time.

Steps 4 and 5 perform the behaviour update part of the process. The only notable point here is that the model checking is only done for those behaviours that have any belief base conditions that are true. Checking these conditions is less expensive than running the model checker, so extra work can be avoided.

Step 7, to select the response to apply requires explanation. The decision to select randomly from the valid responses is mostly borne from a desire to get a working first implementation and time constraints prevented investigation of further options. There are two issues of concern here. Firstly there is a practical issue of agent design and debugging. It is possible that conditions could hold such that applying all the responses could lead to a loop. One expectation could add another which then causes another to apply and so forth. By ensuring only one response can apply each reasoning loop the agent designer can monitor agent execution and more easily debug their design. There is another possible way to avoid this problem that the ESB framework presents - static checks on an agent specification. As the expectation specification is transformed for use with a model checker, systems could be checked for undesirable properties at design time. Checking for possible looping expectation states such as this would involve verifying similar properties to deadlock detection. The possibilities presented for checking agent properties at design time is an area of possible future research.

The second issue here is one that represents an area of possible future work and a theoretical as well as implementation issue. If all valid expectation responses are to be applied and there is more than one – does the order in which they are applied matter? There are two possible options here as to how they are actually applied. It could be the case that one response is applied then further responses are only applied if the expectation containing them is still in the current set. Or it could be that all the responses are applied in some order and those that refer to expectations not in the intermediate state just discarded. This scenario suggests that order might matter,

depending on the exact method of execution. In this case a suitable strategy may be to specify an ordering (e.g. in order of expectation definition) so the designer can achieve the desired effect. The current implementation simply means that the order in which responses are applied is not guaranteed and all responses will be applied if they still hold after the preceding one is applied. It is just the case that it will take multiple reasoning cycles for them all to take effect. This is an advantage for easier agent programming and debugging, for a similar reason to the looping response case.

For an agent designer, this decision means that the test and response application sequence is not defined and so they should create their agent specification with this in mind. An alternative choice could have been to enforce an ordering based on precedence in the agent specification or some other method.

4.3 Discussion

This chapter has taken the ESB theory introduced in Chapter 3 and described the implemented ESB-RS agent reasoning system based on it. The start of the chapter covered the high level design of each component in the system and the second half moved deeper into implementation details. The picture of ESB will be completed by the next chapter, which introduces several case studies and demonstrates the benefits of ESB and, more generally, social reasoning as a separate process.

The use of the NuSMV model checker to test behaviour conditions is the key innovation that allowed for easy implementation. This came from the realisation that the ESB process can be thought of as model generation given a specification, then verifying properties at runtime. The model checker also brings an easy route toward realising some of the benefits that are expected from a generic social reasoning framework, namely analysis of agent designs. As the expectation graph structure can be translated into a form for the model checker, this means even without a specific current state, properties of the expectation graph can be verified at design-time. This is an interesting area for potential future work, as even some brief consideration reveals potential benefits. For example, by using the construct of “eventually” one could check that all expectations are actually accessible, checking that all the reasoning rules are used. Another use could be to check that an agent works through protocols as expected, e.g. checking conditions such that “if the agent expects to be entering a transaction, it will

eventually expect to complete it".

These benefits do not come without limitations however. The use of a model checker for behaviour conditions imposes practical limitations on the language of behaviours and the conditions that can be evaluated. The notable limitation in this case is the inability for behaviours to condition on expectation *content*. The system designer must work with the knowledge of which expectations represent what and write behaviours accordingly. This limitation will be explored in more detail through the case studies in the next chapter.

Strategies have been defined in a very general way, which is both beneficial and a limitation to agent development. This generality means that a great variety of reasoning could be represented, as there are very few constraints, but the downside is that there is less opportunity for general methods of strategy analysis using ESB. This can be contrasted with the behaviours, where a more restrictive language brings generally applicable benefits that are independent of a particular design.

The split of behaviours between the social reasoner and the practical reasoner (to actually carry out actions) is slightly undesirable from a conceptual viewpoint, but has implementation benefits. Principally it allows an agent designer great flexibility in what reasoning goes where. The choice made was for actions to be carried out directly by the practical reasoning component.

The choice of Jason for the BDI part in some way influenced the action implementation. The design of the reasoner was not affected, but it made some components a lot easier to implement. For example, the extension to include variables was easy due to features Jason provides. Due to this the definition of the specifications in XML are fairly generic but certain parts are tied to Jason. The reason is that they use syntax borrowed from Jason to make implementation easier by reusing parsing code etc.

Chapter 5

Evaluation

The previous chapters have defined a framework for general social reasoning and presented an implemented agent reasoner based on the theory. The creation of an implementation forms only the first part of the evaluation of the hypothesis, to form a complete evaluation it is necessary to present and discuss some case studies. The first two sections of this chapter present two implemented agent reasoning schemes in the ESB framework, using ESB-RS. To evaluate further claims regarding the suitability for general algorithms for bounding reasoning, the third section presents two possibilities that ESB presents to place limits on agent reasoning. The final section contains a discussion on ESB in general and evaluates the hypothesis presented in the introduction with respect to the case studies here.

The case studies serve two purposes here. One is to evaluate ESB-RS and show that it is a working implementation of the ESB theory, validating the claim that it is possible to create a general purpose social reasoner. Secondly they help evaluate the ESB theory and by extension the proposed benefits of separate general purpose social reasoning.

The purpose of the section on bounding reasoning in ESB is to provide examples for the evaluation of ESB with respect to the claim that a general social reasoning framework provides new ways to bound an agent's social reasoning processes in a generic fashion.

Selection of social reasoning methods for case studies presented some difficulty. To evaluate the implementation as well as the theory ideally required existing approaches described in enough detail for implementation, if not already implemented. As noted in the background literature review there is a scarcity of social reasoning techniques described at a level of abstraction close to implementation. The reason this is a de-

sirable property is that the goal is to capture agent reasoning, rather than methods to design agents to capture social properties without the agent being required to reason about them. Additionally, it was desirable to choose case studies that exhibit different types of reasoning, to strengthen the argument that ESB is indeed general and thus that evaluations of ESB against the initial hypothesis also validate the hypothesis.

The chosen reasoning schemes were Joint Intentions (JI, Cohen and Levesque [1991]) and the Normative Agent reasoning system (NoA [Kollingbaum, 2005]). Joint Intentions will be introduced first. JI describes how an agent reasons about joint commitment to carry out joint tasks with another agent – a basic principle of MAS. By contrast, NoA is not about inter-agent reasoning but rather a method by which an agent can reason about system level properties. NoA is an architecture that allows an agent to take the norms of a system into account in its reasoning to act according to obligations, permissions and prohibitions. It describes a norm-autonomous reasoning agent, with the agent able to choose to obey or ignore norms according to its priorities. These two approaches are representative of different categories of social reasoning methods and so result in varied implementations for evaluation.

5.1 Case Study: Joint Intentions in ESB

This section evaluates ESB's suitability for capturing social reasoning concerns by demonstrating that the key concept of commitment as used in Joint Intention theory can be easily expressed. The ability to establish joint commitment is vital to MAS. Without it agents cannot work together – a principal advantage of MAS. The most basic form of joint commitment is Joint Intentions, where two agents have interlocking commitments to each other to achieve a joint goal.

To provide some contrast with the ESB implementation and allow for fuller discussion of the merits of social reasoning, an implementation of JI solely using BDI is also presented. This can then be compared to the ESB-RS version.

5.1.1 A Protocol for Establishing Joint Intentions

To design an agent following principles of Joint Intentions, it was first necessary to find a suitable description of how agents can achieve a joint commitment. Kumar

et al. [2002] describe the Request Conversation Protocol (RCP) for establishing a joint persistent goal (JPG). The difference between a Joint Intention and a JPG is in the mutual belief throughout executing the action that it is done as a team. So given a pair of agents believing they are doing the action as a team, the request protocol brings about a state of joint intentions between the agents. ESB is ideally suited to capture Joint Intentions, as the JI centres around commitments the agent holds based on its beliefs about expected commitments of the other agent.

There are several primitive components that the protocol is built upon: the individual commitments; persistent goal (PGOAL) and persistent weak achievement goal (PWAG); and the speech acts REQUEST, AGREE, REFUSE and INFORM. The formal definitions can be found in Kumar et al. [2002] and the implementation of these primitives will be described in the next section.

These primitives are all considered as actions the agent can perform, be they communicative or holding an individual commitment. The social reasoning will be separated out into the ESB side of the reasoner and acts on top of these basic components. These components are therefore implemented in the BDI practical reasoner component. The next section describes a complete BDI based implementation of this protocol and then the subsequent section the extension to utilise ESB.

The RCP is a set of communication acts, with associated semantics and mental states that two agents go through to form a JPG. Figure 5.1 shows the communication acts and the states the agents go through. If agent X is requesting an action A of Y, the intuition is simple. X firsts requests an action from Y. X now holds a persistent weak achievement goal (PWAG) toward Y. Y either agrees or refuses, in the "agree" case Y now also holds a PWAG toward X to achieve the goal. There is now a JI. X has a goal for Y to do A, and Y has a goal to do this action relative to X's desire to do so.

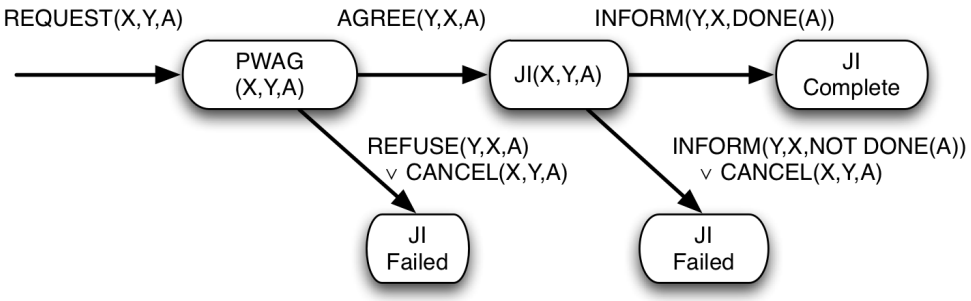

Figure 5.1: The Request Conversation Protocol (adapted from Kumar et al. [2002]).

To simplify the example one assumption is made. We omit the case where the requesting agent cancels the request. This action does not introduce any new states, or new transitions to the protocol, so omitting it simplifies without removing any complexity of reasoning process.

5.1.2 Joint Intentions in BDI

Creating a BDI-only implementation of JI serves two purposes: chiefly that it will allow for comparison with the ESB-RS version where social reasoning is separate from practical reasoning; additionally there is the convenience that it provides an implementation to base the ESB-RS version on, as many of the practical reasoning plans can be reused.

To simplify the implementation and make the core of the joint intentions framework clearer, certain assumptions are made:

Communication always succeeds This is a reasonable assumption to make. JI theory says nothing about ensuring communication and it is a separate problem to that of establishing a joint commitment.

Speech acts always succeed but may not be satisfied. This is slightly different to the above assumption. A speech act is successful when the receiver understands the intent of the message and is satisfied if they act as requested. From a practical standpoint any system of agents using joint intentions created for evaluation is a closed system, so it is easy to ensure by design that the intent of communication is known. This removes the need for the "ATTEMPT" speech acts described in [Kumar et al., 2002] (p189) and increases readability of the implementation.

No Cancel Case As above, this is the case where the requesting agent cancels the request. This doesn't introduce any new states, or new transitions to the protocol, so omitting it simplifies the code.

The following sections build up a description of the complete implementation by first showing how the basic commitment structures are represented, then detailing the various speech acts and how they build upon each other to follow the protocol in Figure 5.1 above.

5.1.2.1 The PGOAL Commitment

A persistent goal is one where the agent wants a goal to become true and will not give up on the goal until it is either accomplished, impossible or has become irrelevant. Conveniently, Jason provides the *Open Minded Commitment* (OMC, Bordini et al. [2007], p174) pattern. This is a pre-processor directive that takes a plan to achieve a goal (make a belief of the same name true) and extends it so that it is single-mindedly pursued until it is achieved, a relevancy condition is no longer believed, or an impossibility condition is believed. This is exactly the definition of a PGOAL. As the OMC pattern must be hard coded, there are several PGOAL cases in the protocol that must be catered for. They are to inform another agent of some fact, to achieve the goal and to achieve the goal relative to a PWAG. These can be general regarding the object they refer to, but still three different instances of PGOAL must be implemented.

5.1.2.2 The PWAG Commitment

The plans to handle creation of a PWAG and the various cases where it is dropped, are the most complex part of the implementation. The implementation follows Kumar et al. [2002], who describe a PWAG informally as:

> "an agent x has a PWAG toward another agent y when the following holds: if agent x believes that p is not currently true then it will have a persistent goal to achieve p, and if it believes p to be either true, or to be impossible, or if it believes the relativizing condition q to be false, then it will have a persistent goal to bring about the corresponding mutual belief with agent y." [Kumar et al., 2002, p184]

An agent X with a PWAG toward Y to achieve P, given Q, with impossibility condition F, can be written as (PWAG X Y P Q). Decomposing this definition, the agent finds itself considering the following situations:

P does not hold and so it has a PGOAL to achieve P.

P holds and so it has a PGOAL to tell Y.

F holds and so it has a PGOAL to tell Y (that P is impossible).

Q becomes false and so it has a PGOAL to tell Y not Q.

It is easiest to build the description of the implementation from the bottom up. The PGOALs to INFORM agent Y of P, F or not Q, are implemented with a generic "tell"

OMC as described for PGOAL above.

The basic operation of the PWAG is implemented in a plan with the following steps:

1. Create the belief that the PWAG is held.
2. Intend and execute a plan to achieve the PGOAL P.
3. Check whether it has been achieved.
4. Intend a PGOAL to tell Y that this is the case.
5. Remove the belief that the PWAG is held.

To fully implement a PWAG, it is necessary to consider some additional complexities. The PGOAL to achieve P may fail initially (and be re-intended, as it is persistent), or the goal may become irrelevant, or impossible. In these cases the above plan will fail and so there must be plans to handle these failure cases (i.e. the above plan fails to execute, not the PWAG).

The possible plan failure cases are handled as follows:

- When it fails because the PGOAL has simply not succeeded, but is still possible and relevant, then the PWAG intention is re-intended. Once this has been done, this intention keeps iterating until the goal succeeds, fails or becomes irrelevant. These cases are handled by individual plans guarded on each termination condition.
- The "failure due to impossibility" case is handled by intending a PGOAL to send an INFORM F message to the other agent (which will cause the other agent to abandon its own PWAG), then dropping the PWAG belief.
- The "failure due to irrelevancy" (Q no longer holds) is handled as per "failure due to impossibility".

For Joint Intentions arising due to the interlocked PWAGs, notification of the various failure or irrelevancy conditions is enough to ensure both agents drop goals as appropriate.

The next subsections consider the different speech acts required to execute the protocol, in the order they are required.

5.1.2.3 INFORM

INFORM is aimed at creating a mutual belief between two agents. Given the assumptions, this can simply be replaced with the standard `tell` performative. This will establish mutual belief under the assumption that communication succeeds. INFORM is an important act, as the others all build upon it.

5.1.2.4 REQUEST

This is the most complex speech act and is the key to the protocol. In terms of the communicative act used, a REQUEST is `tell(Y,request(A,Q,F))` where `Y` is the agent it is sent to, `A` is the action requested, `Q` the relevancy condition, and `F` the condition that indicates the goal is impossible. However a REQUEST is not just the message sent. The meaning associated with it must also be captured.

Assuming agent X is requesting an action of agent Y, a REQUEST involves the following components:

- X holds (PWAG X Y ψ Q) and the request communicates this persistent weak achievement goal so that it is mutually believed.
- The compound goal ψ is that Y performs action A and Y holds a PWAG with respect to X to do A.
- Y's PWAG is relative not only to X's condition Q, but also to X's PWAG.

The plan to issue the request is as follows:

1. Create a PWAG with the composite goal ψ.
2. INFORM agent Y of the request for action A.

And for agent Y receiving a request, the agreement case:

1. Add a belief the other agent X holds a PWAG to achieve the action A.
2. AGREE with agent X the request to perform the action.
3. Create a PWAG to perform the action, with the relevancy condition that is a conjunction of Q and the belief X holds the PWAG added in step 1.

And the refusal case:

1. Send a REFUSE message for the action.

The belief that the other agent holds the PWAG is not sent here, as it is assumed that communication succeeds. Thus it will drop its PWAG as impossible if it is rational (or keep requesting, depending on design).

5.1.2.5 AGREE

The AGREE message is sent by agent Y as above, to the requesting agent X.

The key part of an AGREE is that it informs the receiver, X, that the sender, Y, has a PWAG to do the action, relative to the X's PWAG and relevancy condition. The sender's half is as above, to `tell(X,agree(A,Q,F))`. The receiver (X) must add the belief that Y holds a PWAG (relative to the relevancy condition and agent X's PWAG).

5.1.2.6 REFUSE

A refusal is the case where Y informs that they will never adopt the PWAG to do the action (so of course X's PWAG becomes impossible and must be dropped). A CANCEL would be the counterpart of this issued by X. It is simply implemented as a `tell(X,refuse(A,Q))`, with the Y's half as above. X drops the intention to achieve the action and also the PWAG.

5.1.2.7 Summary

Execution follows the states of the protocol, with the messages exchanged as defined above and the agents moving through the relevant mental states. This is a relatively simple implementation with little agent reasoning and very little of it is what could be called social reasoning. This social reasoning is restricted to the assumptions on the other agent's mental state (the PWAGs) based on the content of the communications. This sounds trivial, and it is a simple example, but it is a good example. It shows how an agent can handle one of the most basic primitives of interaction – commitment. Additionally a simple example is desirable here, to allow explanation with moderate space requirements.

5.1.3 Joint Intentions in ESB-RS

The ESB-RS implementation is based on the BDI only version, as many of the practical reasoning plans are the same. The approach used keeps the primitive actions and goal commitment structures in the practical reasoning component and replaces the social reasoning with an expectation-based representation. It is not suggested that for all ESB designs a BDI (or other practical-reasoner only) version be created first, but was done here to provide additional material for evaluation as it allows comparison between plain BDI and ESB-RS systems.

The plans that are kept as before are; PWAG, PGOAL and the speech acts REQUEST, AGREE, REFUSE and INFORM. These represent (communicative) actions the agent can perform, the social reasoning concerns *when* to take these actions.

Only three expectations are needed to allow the agent to use the above primitives to form and act on a joint intention. The expectations used are shown in Table 5.1. The agent holding each expectation is assumed to be "self" and the other either X or Y depending on the role played. The three expectations can be explained as follows:

1. When X has a PWAG, and expects Y also has the PWAG relative to its own goal, X expects a JI to achieve A exists. The positive test outcome is to believe A (the action was performed), not Q (is no longer relevant) or F (failed). The negative result is harder to express neatly. Loosely it is that A, not Q or F is believed (observed) and this is not communicated by the other agent – they have not upheld their part of the joint commitment. Even if Y does not directly observe the completion event, they should reply when they are informed to ensure mutual belief and that both agents drop the PWAGs.

2. When X receives AGREE(A,Q,F) from Y, X expects Y holds PWAG(Y,X,A,[Q $\wedge$ PWAG(X,Y,A,Q)]). The relevancy condition for Y's PWAG also includes that X still has the commitment toward the goal. Intuitively, it makes no sense for an agent to perform a requested action if the requester is no longer committed to it. The positive test is that Y informs X of A, not Q or F, the negative test is observation of A, not Q or F and that Y does not inform as per 1.

3. When Y receives REQUEST(A,Q,F) from X, expect X holds PWAG(X,Y,A,Q). The tests are as in 2, only Y expects X to inform.

Although these expectations follow logically from the semantics of the communication

acts, they are still *assumptions* about another agent's mental state and so are separated out as social reasoning.

Table 5.1: JI Expectations.
[source(X)] denotes the source of a message as agent X.

	1. ExpJI	2. ExpAgree	3. ExpReq
Condition	pwag(self,Y,A,Q) ∧ pwag(Y,self,A, [Q ∧ pwag(self,Y,A,Q)])	agree(A,Q,F) [source(Y)]	request(A,Q,F) [source(X)]
Φ	ji(self,Y,A)	pwag(Y,self,A, [Q ∧ pwag(self,Y,A,Q)])	pwag(X,self,A,Q)
T+	A ∨ ¬Q ∨ F	(A ∨ ¬Q ∨ F) [source(Y)]	(A ∨ ¬Q ∨ F) [source(X)]
T−	A ∨ ¬Q ∨ F [source(¬Y)]	(A ∨ ¬Q ∨ F) [source(¬Y)]	(A ∨ ¬Q ∨ F) [source(¬X)]
ρ^+	–	–	-
ρ^-	–	remove agree(A,Q,F)	remove request(A,Q,F)

Only a couple of responses are required. In the failure cases, the belief that the other agent has agreed or requested an action is removed. The effect is to remove the expected PWAG from the belief base and thus the expectation of a JI (if any). In addition, it may be desirable to add some response actions as a result of JI being upheld or not. An obvious case would be maintaining a list of other agents who have been proven willing to cooperate, to aid future decisions.

In this simple set of expectations, there is no real need to choose a strategy, as there are not significant numbers of transitions in the graph. Strategies are envisaged as becoming more useful to bound and direct reasoning in more complex situated examples. Strictly this example can be thought of as using an "unbounded" strategy.

Only one behaviour is required for an agent to act upon joint intentions, it is:

ji(self,Y,A) ▷ *add_belief*(haveJI(A))

This says that when an agent expects to have a JI to do an action A, it adds the synthetic belief *haveJI(A)* to its belief base. This belief represents the JI and should act as a guard on all plans that require the JI to succeed. So it can be read simply as, "allow joint plans

only when believing a joint intention is expected". This seems very obvious and simple but this is the nature of Joint Intentions. It is the most basic building block on top of which more complex social reasoning and interaction can be built. Here it is specified explicitly, rather than designed into a system implicitly.

Although this is all that is necessary to express joint intentions to perform the same as the BDI version, ESB provides an easy route to generically answer the social reasoning questions of when to request a joint action and when to agree to one. Both are necessary considerations for any implementation and help demonstrate how ESB can help bridge the gap from theory to implementation.

The first consideration is when to request a JI. Specifically, when it is possible in the future to hold a JI toward A, and it is desired, then request:

$$\mathbf{E}\Diamond(\text{ji(self,Y,A)}) \wedge (\textbf{desire}\ \text{A}) \rhd \textit{add\_belief}(\text{self requestJI(A)})$$

The condition is similar for agreeing to a request, the only difference being the requirement that a request has been received:

$$\text{request(A,Q,F)} \wedge \mathbf{E}\Diamond(\text{ji(self,Y,A)} \wedge (\textbf{desire}\ \text{A}) \rhd \textit{add\_belief}(\text{agreeRequest(A)})$$

In terms of expectation graphs, the condition says that if from the current state, a possible future expectation state includes one in which I hold a JI – then it is sensible to request (or agree) to one. The converse is that it is not rational to request or agree to a JI if an agent doesn't believe that its future reasoning (or the other agent's) can bring about this state.

5.1.4 An Extended Example

To evaluate how ESB can be easily used in practice and extended in a modular way, this section presents a robotic soccer example using the JI protocol described above. The situation we consider here is an example of a team plan to score a goal – thus requiring joint commitment between agents, as presented above. The scenario involves three team-mates, one with the ball. The agent in possession of the ball wishes one agent to go up either side of the pitch, one as a feint and the other to receive a pass, then shoot.

If the agent with the ball is Andy, wanting to pass to Barney and have Cathy perform the feint, the intended operation is as follows. Andy must form a JI with Barney to move up the pitch receive a pass and shoot and with Cathy to move up the pitch. But

it can also be assumed that each agent also has practical reasoner plans to score goals a variety of ways, in different situations. To handle when to request and agree to joint actions in this example, the set of expectations presented previously is supplemented with two expectations as follows:

ExpFree When another agent does not have the ball, expect that they are "free" (to accept requests). The test for this is if they accept or refuse a request. The positive response is to add ExpJI and ExpAgree from Table 5.1 to the current set, and remove ExpTeam.

ExpTeam When another agent has the ball, expect it to desire a joint intention for some team action to score. The test is if they request some joint action. The positive response is to add ExpJI and ExpReq from Table 5.1 to the current set, and remove ExpFree.

This produces the expectation graph shown in Figure 5.2. The basic idea of these expectations is to demonstrate one of ESB's strengths, bounding reasoning. By only adding the expectations about joint intentions to the current set when they become applicable, it reduces the number of conditions and expectations that must be maintained. This also provides a very natural way to represent any scenario where an agent may act in different "scenes" performing different roles. Also simply by extending the above general JI expectations, in a modular fashion, we get this benefit. So it is easy to add and integrate reasoning rules together.

Figure 5.2: Expectation Graph for the Robosoccer Agents. Edges are annotated with the responses defining the state transitions.

There is now enough of the example defined to demonstrate another advantage of ESB. It is possible to easily perform checks on sets of expectations to aid the designer. For example, in this case the intention is that an agent should only ever eventually either be in a state to expect to receive a request for a JI, or to try and form one. By exploiting the automatically generated FSM, we can query the model checker at design time and check if the set of expectations meets our design. In this case we test to ensure it's

always the case that both expectations do not hold together:

$$\mathbf{A}\Box\neg(ExpAgree \wedge ExpReq) \tag{5.1}$$

If this test should fail, the model checker provides an error trace showing exactly the states and transitions leading to the failure and the design can be corrected accordingly.

As well as implementation specific tests, it is also possible to describe general tests that the ESB formalism allows and that can be applied to any design. It is reasonable to expect that if there is an expectation which can never hold, then there is a problem with the design. The following condition can be checked for each expectation:

$$\mathbf{E}\Diamond(Expectation) \tag{5.2}$$

In this example, if it were broken by making the response of *ExpTeam* empty, then this test would fail for *ExpReq*. The design could be fixed through manual inspection of the expectations, or by identifying more specifically when the behaviour is not as intended by checking a more complex property and inspecting the error trace.

Behaviours are as easily extended as the expectations. As before, there is a behaviour to add a belief noting when the agent holds a joint intention, so that BDI plans can take advantage of this. When it is expected that a JI is held, plans depending on it are allowed via a guard belief representing this. The behaviours presented above dictate when an agents expects it is possible to form a JI can be included as before, or for convenience slightly extended as follows:

$\mathbf{E}\Diamond$(ji(self,Y,A)) $\wedge$ free(Y)$\rhd$ *add_belief*(requestJI(Y,A))

request(A,Q,F) $\wedge$ $\mathbf{E}\Diamond$(ji(self,Y,A)) $\wedge$ free(self) $\rhd$ *add_belief*(agreeRequest(A))

Added here are the requirements on the expectations that relate to agents being free to accept JIs and only agreeing to a JI if an agent is itself free. The intuition is that if an agent has the ball, it will have better options for scoring itself and will not agree to a proposed joint action (though it may propose one itself). As the first behaviour uses the temporal operators, even though an agent initially is in a state where no expectations about JIs are current, the condition takes into account possible future reasoning and so the behaviour holds.

The final component of the soccer agents is the set of BDI plans. In AgentSpeak(L) plans are of the form:

```
+!goal : guard
 <- actions or subgoals.
```

Plans are selected on a priority ordering from the set of relevant plans – those whose guard holds and trigger meets the current goal. They then become an intention, and for each cycle one intention is chosen to execute the next step from. Steps in a plan may be actions or themselves subgoals, triggering a further plan selection process.

Amongst other plans to achieve the goal `score`, the social reasoning agents have the following plans:

```
+!score : haveBall & ji(self,Barney,Pass) & ji(self,Cathy,Feint)
   <- passTo(Barney).
+!score: request(Andy,Action) & agreeRequest(Action)
   <- .tell(Andy,agree(Action)).
+!score : ji(self,Andy,Action)
   <- do(Action).
+!score : haveBall & requestJI(Barney,Cathy)
   <- request(Barney,Pass);
      request(Cathy,Feint).
```

The beliefs underlined are synthetic ones added by behaviours and represent the interface between the ESB social reasoner and BDI practical reasoner. Using beliefs added by the ESB reasoner as guards on plans helps bound the agent's reasoning in another important way – it is only necessary to consider plans consistent with the agent's current social reasoning.

5.1.4.1 Execution of the Soccer Agents

Evaluation of the JI implementation is by examining the execution of the ESB-RS agents to ensure that they move through the correct mental states via the appropriate messages. To show that this is the case and help illustrate the implementation described in this section, the key states in the execution of the extended example will be described and compared to the RCP theory.

To better illustrate the example scenario, Figures 5.3 and 5.4 show the four key stages of the extended soccer example.

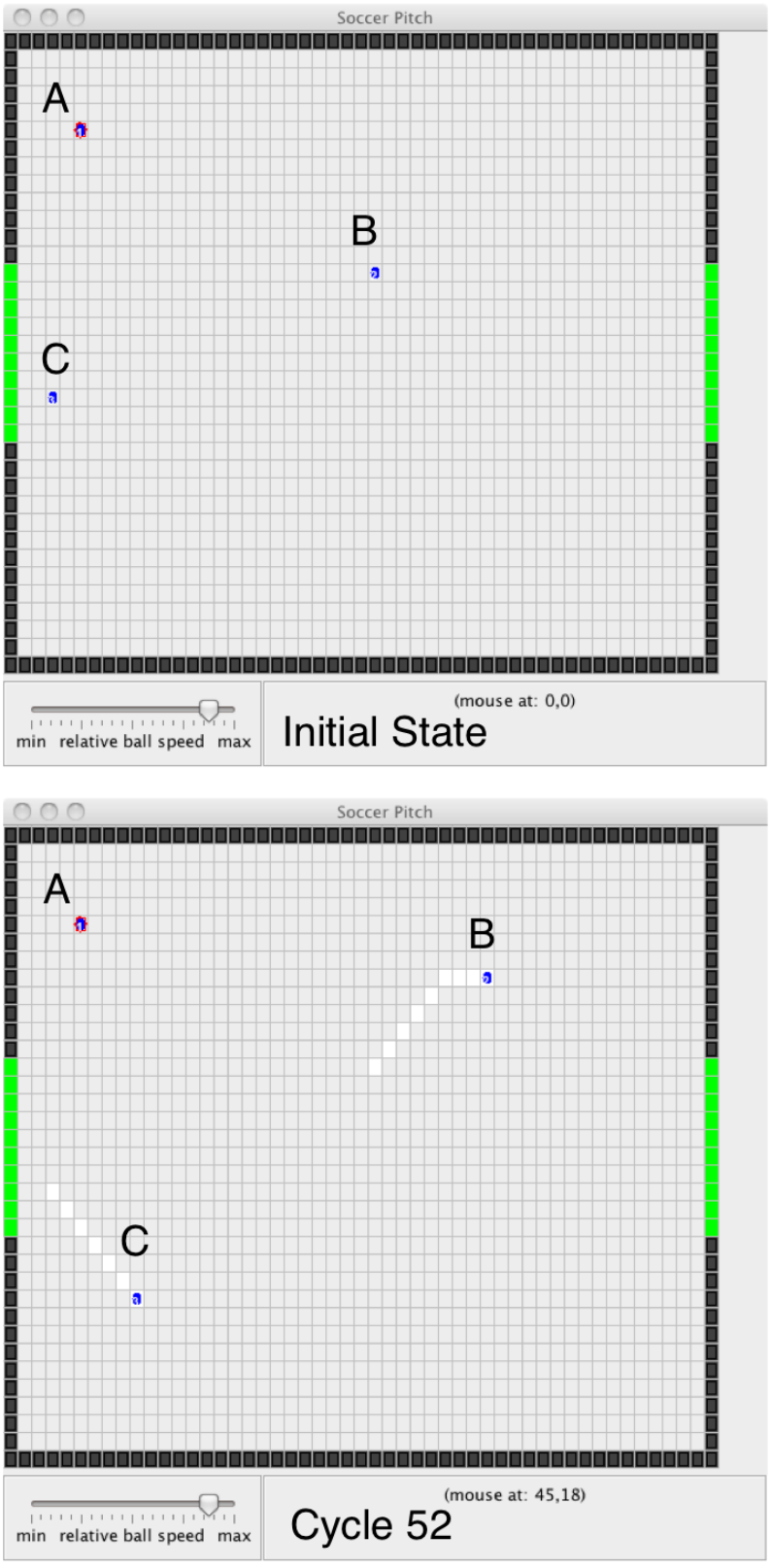

Figure 5.3: The initial state and the two agents accepting and carrying out their part of the joint action.

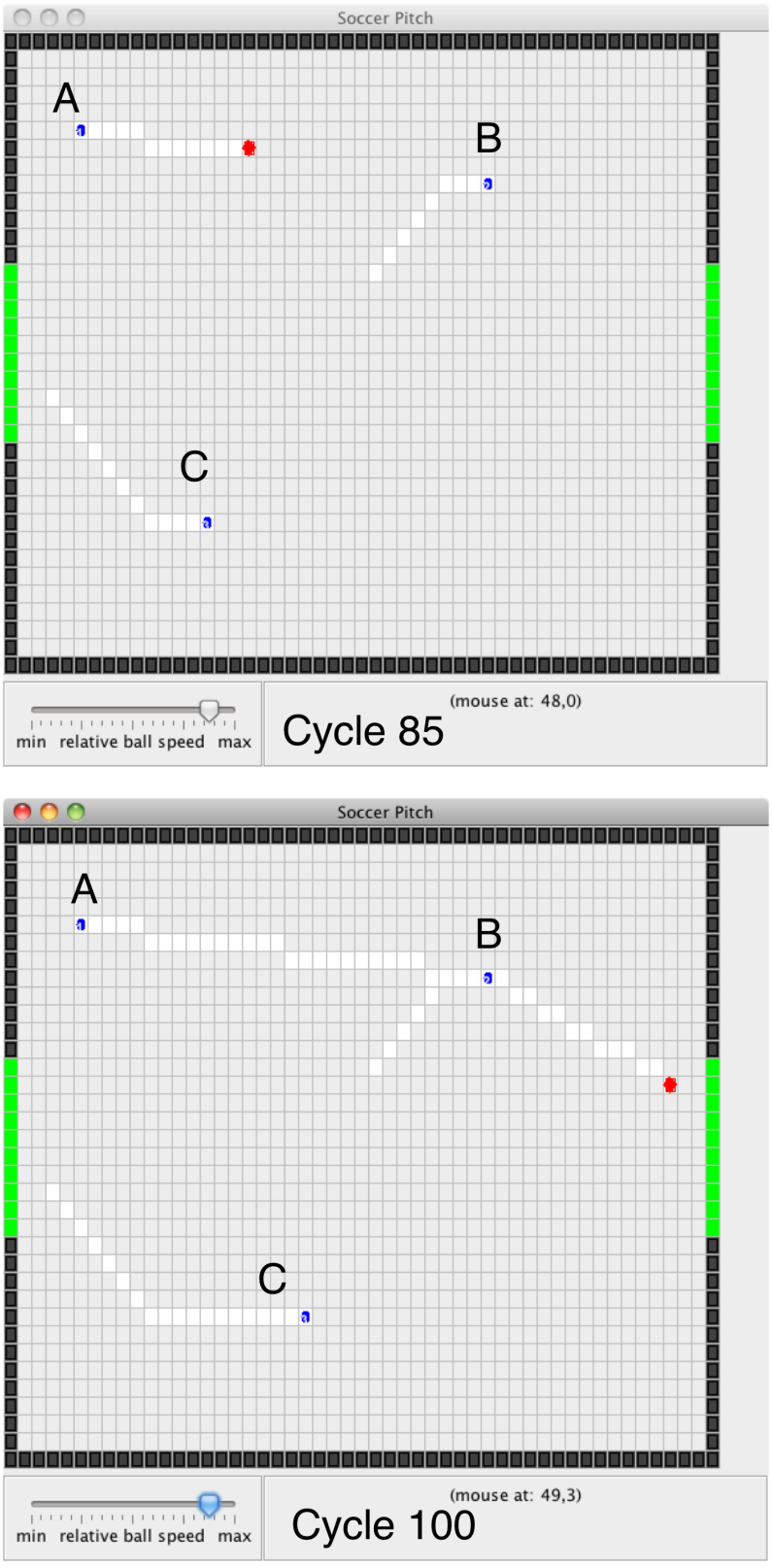

Figure 5.4: Agent A passes the ball to B in the second key step, and B shoots whilst C continues to feint.

To show each cycle of the agents' reasoning processes would take many pages, so only the key steps will be shown. Similarly all of the agents' beliefs and intentions are not shown, only those relevant to key steps. Initially the agents find themselves in state shown in Figure 5.3 with the following mental states:

Step 0
Andy

$EXP_A(\mathbf{EXP_C})$	Beliefs	Current Plans
expFree, expTeam	**free([cathy,barney])** requestJI([cathy,barney])	Send request to to barney and cathy then resume goal to score.

Barney

$EXP_A(\mathbf{EXP_C})$	Beliefs	Current Plans
expFree, **expTeam**	**free([cathy])** **desireJI(andy)** requestJI([cathy])	wait for a request

Cathy

$EXP_A(\mathbf{EXP_C})$	Beliefs	Current Plans
expFree, **expTeam**	**free([barney])** **desireJI(andy)** requestJI([barney])	wait for a request

The tables can be read as follows. The first column shows the agent's active expectations with those current ($C = true$) in **bold**. The second column shows key beliefs the agent holds, with **bold** indicating the source is an expectation and underlining indicating the source is a behaviour. The final column provides details of the content of the relevant intentions the agent currently holds. The example will proceed with descriptions of the agent mental states in this style with brief explanatory text between to show the key message exchanges and relate to the stages of the JI protocol.

This initial state represents the entry point to the request conversation protocol, as introduced in Figure 5.1. *expFree* is current for all agents and as such they have beliefs added with the lists of agents they observe as free. Consequently the behaviours that say when to request a JI have added this to the beliefs to represent these agents might be open to a joint action. The only joint action plan the agents possess requires two other agents, so it is only Andy intending a plan – to issue a REQUEST communication act to Barney and Cathy, which he does. Barney and Cathy both desire this JI, as per their *expTeam* expectation. This brings the agents to the next step, where requests have

been received and the various PWAGs are held:

Step 1
Andy

EXP_A(**EXP_C**)	Beliefs	Current Plans
expFree, expTeam	**free([cathy,barney])** requestJI([cathy,barney]) PWAG(andy,barney,pass) PWAG(andy,cathy,feint)	Waiting for responses to the REQUEST speech acts.

Barney

EXP_A(**EXP_C**)	Beliefs	Current Plans
expRequest, **expTeam** expJI	**desireJI(andy)** requestJI([cathy]) agreeReq(pass(andy)) **PWAG(andy,barney,pass)**	Send agreement to Andy, then carry out move to receive pass and shoot.

Cathy

EXP_A(**EXP_C**)	Beliefs	Current Plans
expRequest, **expTeam** expJI	**desireJI(andy)** requestJI([barney]) agreeReq(feint(cathy)) **PWAG(andy,cathy,feint)**	Send agreement to Andy, then carry out move to feint.

This represents the mental states for the first node in the RCP diagram in Figure 5.1. Andy has adopted the two PWAGs towards Barney and Cathy and is now waiting for a response to the requests (without which they would be abandoned). The *expFree* expectation is no longer active for Barney or Cathy as *expTeam* had the positive test confirmed, the response to which was to remove the expectation and add *expRequest* and *expJI*. Only *expRequest* is current of these two and the PWAGs Barney and Cathy believe Andy to hold are added by this. Additionally the behaviour stating when to agree to a request is now true, as the model checker shows that there is a possible future state where *expJI* can hold and a request has been agreed. So Barney and Cathy plan to send an AGREE communication and carry out their plans. This communication, once sent, will generate PWAGs for Barney and Cathy toward their joint actions with Andy and so they will expect they have a JI. Andy will receive the AGREE and change expectation state. This then transitions the agents to the second *JI(X,Y)* state of Figure 5.1. Their internal state is therefore as follows:

Step 2

Andy

EXP_A(**EXP_C**)	Beliefs	Current Plans
expFree, expJI, expReceive	**free([cathy,barney])** PWAG(andy,barney,pass) PWAG(andy,cathy,feint) **PWAG(barney,andy,pass)** **PWAG(cathy,andy,feint)** **ji(andy,barney,pass)** **ji(andy,cathy,feint)** haveJI(andy,barney,pass) haveJI(andy,cathy,feint)	Waiting for Barney to reach position to receive the pass, plan triggered by the haveJI behaviour action.

Barney

EXP_A(**EXP_C**)	Beliefs	Current Plans
expRequest, **expTeam** **expJI**	**desireJI(andy)** **PWAG(andy,barney,pass)** PWAG(barney,andy,pass) **ji(barney,andy,pass)** haveJI(andy,barney,pass)	Moving to position to receive pass and shoot at goal.

Cathy

EXP_A(**EXP_C**)	Beliefs	Current Plans
expRequest, **expTeam** **expJI**	**desireJI(andy)** **PWAG(andy,cathy,feint)** PWAG(cathy,andy,feint) **ji(cathy,andy,feint)** haveJI(andy,cathy,feint)	Move up one side of the pitch in a feint move.

Andy, Barney and Cathy now expect they have interlocking PWAGs with each other and so expect a JI exists (*expJI* holds). This causes the relevant behaviours to add the "haveJI" beliefs that act as guards on their various plans to carry out the actions they have committed to with Andy. The agents then carry out their parts of the plan as illustrated in Figures 5.3, 5.4. When Barney finally scores, shortly after the 100th reasoning cycle, the agents discharge their JIs by sending INFORM messages that the goals are complete. This leaves the agents in the following mental states, which match the final "JI Complete" state in Figure 5.1.

Step 3
Andy

EXP_A(**EXP_C**)	Beliefs	Current Plans
expFree, expJI**, expReceive**	**free([cathy,barney])** pass(andy,barney)[from(barney)] feint(cathy)[from(cathy)]	-

Barney

EXP_A(**EXP_C**)	Beliefs	Current Plans
expRequest, expTeam expJI	pass(andy,barney)	-

Cathy

EXP_A(**EXP_C**)	Beliefs	Current Plans
expRequest, expTeam expJI	feint(cathy)	-

Here all of Barney and Cathy's expectations are no longer current and Andy is back to holding only *expFree* and *expReceive* (which could have been removed also, depending on design). This is because the actions specified in the JI and interlocked PWAGs were carried out. Barney and Cathy therefore dropped their PWAGs and informed Andy, who then dropped his PWAG. This makes the *expJI* expectation no longer current and the expectations are removed.

5.1.5 Evaluation

The purpose of creating an ESB-RS implementation of JI was to aid evaluation of both the ESB-RS implementation and the conceptual ESB framework.

A separate issue to evaluation of the ESB framework is evaluation of the quality of this JI implementation in its own right – does it accurately reproduce JI reasoning. One avenue to show this would be formal proofs. As the semantics of JI are well studied, given a semantics of ESB, Jason and the various speech acts, this could perhaps be shown. This would be a significant undertaking and principally a study of the AgentSpeak(L) semantics, possibly also requiring additional work to further formalise the description of ESB. As stated in Chapter 3 this is something that was not tackled for

this thesis; the focus is on showing a complete ESB theory and implementation.

The principal means of evaluating the implementation was simply to compare the agents' mental states using the debugger as they proceeded through the protocol. These states can then be compared to the described protocol and the agents confirmed to be exchanging the correct messages, resulting in the correct beliefs being inferred.

The extension to the robosoccer example of course cannot be checked in this way as it is a new implementation created to test ESB. It is possible to see that the JIs are formed and discharged in the same way and that it performs the task as intended. The various cases for failure can be checked. If one agent refuses or finds their part of the plan impossible, the other agents drop their intentions toward the plan. This is the essence of JI. It allows agents to perform an action only when the joint commitment of a team of agents will ensure it succeeds.

The JI implementation was straightforward from the ESB point of view. Most of the complexity in the code is implementing the basic communication constructs in the practical reasoner. This was expected, JI represents a very basic form of social reasoning so it is appropriate that it is easy to capture with ESB. The extended example illustrates a big strength of ESB, modularity and ease of extension. This is not necessarily a benefit of all forms of separate social reasoning but it is clearly a possible advantage as shown here.

5.1.5.1 ESB-RS Compared to Regular BDI

Agents to follow a generic JI protocol were created both in ESB-RS and purely in Jason. This allows for a limited comparison between a traditional practical reasoner only approach and separate social reasoning. The comparison is limited as there is not much of the JI process that can be considered as pure social reasoning, the bulk of the effort was in creating the communicative acts, rather than the selection of when to carry out the acts. This does help answer the question of where the divide between practical and social reasoning should be, as it was necessary to consider this in creating the ESB-RS version from the plain BDI version. The decision was that any plans to carry out actions, including communication actions, would be left in the practical reasoner. This makes sense as the social reasoner cannot directly cause the agent to carry out an action. The social reasoning part is therefore when to communicate in this case. The RCP protocol to establish a JI is fairly restrictive, so there is no great difficulty in

encapsulating this purely in BDI.

Maintaining the PWAG commitments is part of the practical reasoning half of the agent. At first this may seem counter-intuitive, as commitments may appear a social construct. This is certainly the case for the joint commitments, which *are* captured in ESB. The one-sided PWAG structures are most closely related to the open minded commitment an agent may hold toward its goals and so this is how they are represented. Acting on these commitments and basing a joint commitment upon them is carried out in the ESB component.

To reiterate what has been written earlier, the main benefit seen by the ESB-RS version was easy extensibility. This is due to the modular nature of the ESB specification. Section 5.1.4 shows how it is trivial to add additional expectations, without changing existing expectations, that reason about when to attempt to form a JI and accept requests. It is easy to see how this could be extended to combining social reasoning schemes. For example, agents could jointly commit to following certain norm specifications. This would be an interesting area of potential future work, it is just a matter of implementation. It is made possible by ESB and separating social reasoning from practical.

5.2 Case Study: Norms in ESB

The second case study was chosen to represent a different style of social reasoning. Whereas Joint Intention principles describe an agent's reasoning about the intentions of another agent, norms require an agent to include the rules of the social system in which it operates into its reasoning. As shown by the background study, system level approaches to social reasoning such as norms are usually used with the goal of restricting reasoning to ease the burden of designing an agent that can function in a society. In contrast, the example here is of a norm autonomous agent (see Section 2.1.2.3.3) which must reason about norms and decide to obey them or not. It must also take into account the effects of norms on possible plans.

5.2.1 NoA – Norm Governed Practical Reasoning

The normative reasoning system chosen for implementation was NoA, Normative Agent architecture [Kollingbaum, 2005]. NoA describes a model and implementation for agents to account for norms in their practical reasoning. The goal for evaluation is not to exactly copy the implementation and reasoning algorithms used by NoA, but to capture the key properties of the NoA model in ESB. To support the argument that ESB is generic social reasoning a transformation from a NoA norm specification to ESB is shown. This is tested with two examples from Kollingbaum [2005] translated into an ESB-RS implementation.

Not all of the NoA reasoning architecture is captured here, to recreate the entirety of the work would be an over-ambitious goal. The focus is on the parts of the NoA system that model how norms motivate and restrict an agent's action choice.

The next section will describe the NoA model in more detail, but remaining at a high level there are a few key properties of the NoA system:

- Allows an agent to reason over three key types of norm:

  Obligations which motivate an agent to achieve certain states or perform actions.

  Permissions which explicitly allow actions or states.

  Prohibitions that deny certain states or actions.

- Allows for norm-autonomous behaviour, an agent can deliberate over which norms to adopt.

- Allows for norms to refer to states of the world and particular plans or actions.

- Detection of conflicts, to allow for deliberation over conflicting prohibitions and permissions.

- Detection of inconsistencies when adopting obligations. This is the case where an obligation motivates an agent to act inconsistently with its norms.

These are the key benefits to agent reasoning that NoA brings and it is these properties that the ESB design also holds.

5.2.2 The NoA Reasoning Model

To fully describe the NoA framework here would be inappropriate and space-consuming, so only an overview of the salient points is presented. More details are provided where necessary in the next section, describing how it is captured with ESB.

At a high level there are several characteristics that define how a NoA agent reasons. Obligations are the source of the agent's motivation to act and drive plan selection. Permissions and prohibitions restrict the set of plans, and plan instantiations, that are consistent. Plans are specified in terms of the action they carry out and their effects. This allows for norms to reference both states and actions.

As well as the usual set of beliefs and plans, a NoA agent also maintains a set of norms and roles. Norms are held by a particular role and an agent may hold several roles.

5.2.2.1 Norms in NoA

A norm has the following components:

Label – Obligation, prohibition or permission.

Role – The role the norm applies to.

Activity Specification – Describes either a state of affairs or action to perform (or its negation).

Activation Condition and Expiration Condition – These define exactly when the norm applies.

Not all combinations of norms and activity specifications are supported by NoA directly; many are re-written. For example:

> *¬permission(¬achieve(p))*
> "not permitted to not achieve p"

can be rewritten as:

> *obligation(achieve(p))*
> "obligated to achieve p"

Certain norms have no effect on an agent's reasoning, e.g. "*permission(¬achieve(p))*".

The separate activation and expiration conditions allow very fine control over the scope of norms. As soon as the activation holds, the norm becomes current and stays current

until the expiration condition (regardless of the state of the activation condition). Activation conditions may contain variables, in which case the action specification is bound to possible values for these. In this way a single norm specification may represent a set of norms.

Of less importance here, as they will not be presented in the ESB version, are contracts. These describe a set of agent to role mappings and the norms for those roles. They are not considered here as they have no direct impact on the social reasoning and plan selection process. The principal influence of contracts is that they require an agent to deliberate about norm adoption namely whether it should agree to a contract or not. This is still social reasoning, but not covered here. To reiterate, it is not the goal to reproduce all of the NoA work rather to just show that it is possible to capture the method of normative reasoning in ESB.

5.2.2.2 Plans in NoA

Plans in NoA are selected in one of two ways and their selection is motivated by the action specification of adopted obligations. Explicit effect specifications added to plans allow for selecting plans according to a norm specifying desired state of affairs. If a norm specifies an action, plans are selected according to their name and parameters - the signature. Much like norms, plans can also have variables which are bound when they are instantiated, so it may be the case that only certain instantiations of a plan satisfy the norms an agent holds.

A plan in NoA has the following components:

Signature – this is composed of the name and parameters of the plan.

Precondition – is a condition on the agent's beliefs.

Effects – are a list of the effects that occur during plan execution.

Body – is the executable part of a plan.

5.2.2.3 Reasoning in NoA

This section describes the reasoning process of a NoA agent. The focus is on the properties of the reasoning system, and how the norms are processed, rather than on the details of the NoA implementation.

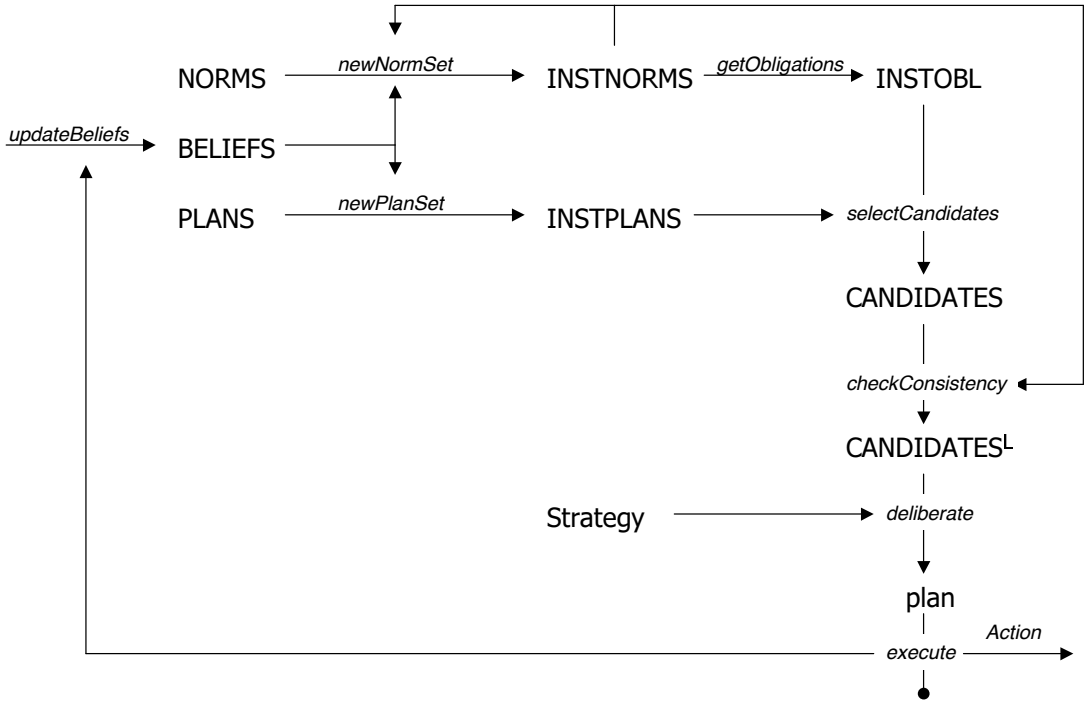

Figure 5.5: Dataflow within the NoA Model (diagram reproduced from [Kollingbaum, 2005]).

Figure 5.5 shows the dataflow of the NoA reasoning cycle and the various sets the agent maintains are processed in the cycle as follows:

1. Update BELIEFS.
2. Instantiate active norms to form INSTNORMS.
3. Instantiate all plans relevant to current beliefs to get the set INSTPLANS.
4. From INSTNORMS create the set of obligations, INSTOBL.
5. Generate the set CANDIDATES, which is the set of all instantiated plans that achieve the obligations INSTOBL.
6. Label each plan in this set consistent or inconsistent with the prohibitions and permissions to create CANDIDATES^L.
7. Using some deliberation strategy select a plan from CANDIDATES^L to execute.

There are a few details of this process worth expanding on to support future discussion. Firstly, plans can have multiple instantiations. This is important as it may be the case that only certain instantiations will be consistent with the agent's norms, either in the

action performed or indirectly through effects. Similarly norms may also be instantiated in several ways depending on possible unifications for their activation conditions. Obligations are separated out from the other norms, as in the NoA system these are the motivators for action, similar to intentions in a classical BDI reasoning process.

The set of candidate plans also allows for multiple instantiations. This step is performed before consistency checking and it may be that certain instantiations satisfy some norms and not others.

NoA classifies the consistency of the set INSTNORMS into three states, namely; strong consistency where all candidate plans are executable without violations, weak consistency where there are at least some consistent plans and strong inconsistency where no plan is consistent with the norms. Labelling the norms with consistency is an important step, as it allows deliberation over all norms, creating a truly norm-autonomous agent that may choose to ignore them. An interesting observation here is that NoA does not propose any strategy to deliberate about plan selection in the presence of norm violations, that is left to the agent designer.

Two very important concepts in NoA are those of conflict and consistency. Consistency refers to plans and obligations. This is the issue of plans being consistent with norms and new obligations being consistent with existing norms. It could be the case that the plans motivated by a new obligation violate existing norms. Conflict occurs where prohibitions and permissions clash and more than one norm affects the validity of an action or state of affairs. NoA requires that all conflicts are resolved in the reasoning process, but inconsistencies may not be. As above, a norm autonomous agent must be able to deliberate over these.

Consistency is the simpler concept. It requires checking that a specific plan instantiation is allowed as an action and also that the effects are permitted (unified as appropriate).

Regarding conflicts there are two key issues, these are detection and resolution. Three properties of NoA allow for detection: preconditions, explicit effects and activation / expiration conditions. Given these properties and a couple of definitions, the conflict resolution strategy of NoA can be summarised. There are three general types of conflict that can occur, considering possibly partially instantiated norms. Two conflicting norms may have the same scope. This is the case, for example, if one norm was to permit a state *on(a,b)* and another was to forbid it.

The overlap case occurs when one norm pertains to a subset of another's states. So for example, an agent may have a prohibition $\forall X, Y$: *on(X,Y)* and then a more specific permission *on(a,b)*.

The scope of two norms may also intersect, where some possible plan instantiations could be subject to both norms but one norm is not strictly more specific than the other. An example would be one norm permitting $\forall X$: *on(X,b)* and another prohibiting $\forall Y$: *on(a,Y)*. In this case there is conflict for the effect *on(a,b)* but there are states where only one applies, e.g. *on(c,b)*, *on(a,c)*.

Given these definitions the NoA conflict resolution strategy is as follows:

- Agents are by default in a permissive state, all states and actions are allowed.
- Any norm explicitly adopted overrides this default.
- Where two norms share the same scope, the more recent dominates.
- Where there is overlap and one norm is strictly more specific in scope, the more specific norm takes preference over the more general.
- As norms are activated they are added to a stack, when they expire they are removed. So a specific permission may expire leaving a more general prohibition.
- In the case of intersection no specific strategy is mandated, though there are several suggested.

The intersection case is the most complex as there is no clear cut way to resolve such cases. Several options are suggested, such as obeying the most recent norm, a bold agent that always follows permissions and other similar tactics.

With this brief overview of the principal properties of the NoA reasoning model an ESB-RS implementation can be described in the next section.

5.2.3 NoA in ESB-RS

The blocks world example from [Kollingbaum, 2005, p69] is used to illustrate how NoA is implemented in ESB-RS. After the overview of the specification a more detailed letter-of-credit protocol example will be presented. This section will build up the components necessary to capture NoA properties. Implementation details are not

covered here, as the intricacies of the code have no bearing on the evaluation of ESB as a framework.

Norms are at the heart of NoA, so it is natural to begin with a description of how norms are translated into expectations. The example features the following norms for a block moving robot:

```
obligation (
    robot,
    achieve clear ( X ),
    not clear ( X ),
    clear ( X )
)
```

```
prohibition (
    robot,
    achieve on ( X, ''c'' ),
    TRUE,
    FALSE
)
```

The first term in the norm specification states the role this norm applies to – in these examples it is the `robot`. Next is the action specification – a state to achieve or action to perform. Finally the activation and expiration conditions are specified.

Each norm is translated into a pair of expectations, one to capture the details of the norm and a complement expectation to handle activation and deactivation. The intuition is that the agent's current expectations represent the norms currently held. Active expectations represent norms that the agent has adopted where the activation condition is not yet met. A norm captured in an expectation can roughly be read as "when the activation condition holds, expect the norms action specification, when the expiration condition holds remove this expectation.". This gives a hint to the purpose of the tests. The test of the norm expectation is the expiration condition, when it holds the norm expectation is removed from the active set and the complement replaces it. The complement is conditioned on the activation condition and immediately applies the response to add the norm expectation. The expectations for the two norms above are shown in Table 5.2 and 5.3

Note that there is no complement expectation required for *ProhibitionOn* as it is defined as being always active (or rather there is, but it is never active).

These examples highlight quite a few key considerations for translating NoA norm specifications into ESB. Firstly, it would seem that repeating the activation condition in *ObligationClear* is unnecessary as it forms a disjunction with true so the condition always holds. This is to ensure that norms that should be active are current expectations, and the condition is repeated so that any variables used in the action specification

Table 5.2: Obligation-to-Clear Expectations

Expectation	ObligationClear	oClearComp
Condition	(not clear(X) $\wedge$ not relaxOC) $\vee$ true	(not clear(X) $\wedge$ not relaxOC)
Φ	achieve(clear(X))	true
T+	clear(X)	true
T-	false	false
ρ^+	add(oClearComp), remove(ObligationClear)	add(ObligationClear), remove(oClearComp)
ρ^-	none	none

Table 5.3: Prohibit-On Expectation

Expectation	ProhibitionOn
Condition	not (relaxPO)
Φ	$\neg$allowed(on(X,c))
T+	false
T-	false
ρ^+	none
ρ^-	none

are bound. The complement expectation *oClearComp* is trivial. The condition is the activation condition and the response applies immediately making *ObligationClear* active. Similarly $T+$ for *ObligationClear* is simply the expiration condition and ρ^+ replaces the norm expectation with the complement. $T-$ and ρ^- are not required to capture norms. To reiterate, ESB aims to be general, so it is not the case that all reasoning schemes will require all parts of the machinery.

The action specifications captured in Φ require more detailed explanation. Although the name of the expectation describes whether it is an obligation, prohibition or permission, this is just cosmetic to aid readability. The true definition of each norm is in the form of the Φ specification. Obligations are specified in terms of the goals they motivate - achievement or performance of a state or action. Prohibitions and permissions are in terms of actions or states that are allowed or disallowed. This is summarised in Table 5.4 which shows for each type of norm how it is captured.

The various forms of action specifications and how they contribute to plan selection re-

Table 5.4: Translation of Action Specifications into ESB Representation.
P represents a state of affairs, X an action.

Norm	ESB Φ Belief
permission(achieve(P))	`allowed(P)`
permission(¬achieve(P))	Not used in NoA.
permission(achieve(¬P))	`allowed(notA(P))`
permission(¬achieve(¬P))	Not used in NoA.
prohibit(achieve(P))	`¬allowed(P)`
prohibit(¬achieve(P))	obliged to achieve(P)
prohibit(achieve(¬P))	`¬allowed(notA(P))`
prohibit(¬achieve(¬P))	obliged to achieve ¬P
obligation(achieve(P))	select a plan with allowed(p) in effects.
obligation(achieve(¬P))	select a plan with allowed(notA(p)) in effects.
obligation(¬achieve(P))	prohibit achieve(P)
obligation(¬achieve(¬P))	prohibit achieve(¬P)
permission(perform(X))	`allowed(X)`
permission(¬perform(X))	Not used in NoA.
prohibit(perform(X))	`¬allowed(X)`
prohibit(¬perform(X))	obliged to perform(X)
obligation(perform(X))	select plan X
obligation(¬perform(X))	prohibit perform(X)

quire some explanation. The obligations are special cases, which motivate plan selection. The prohibitions and permissions add a (possibly partially instantiated) predicate to an agent's beliefs that determine what states and actions are permitted. A point of note is that "not" is explicitly specified, as this is necessary for conflict resolution between norms, which will be explained shortly. For states of affairs, an explicit `notA(P)` predicate is used. This is required for implementation purposes. The reason for this is that it is not required to check if P is in the belief base (i.e. ¬P is true), but rather the predicate is used when matching effect specifications of plans. `notA(P)` in a plan effect specification indicates explicitly that not P is achieved as an effect.

An important part of NoA is that plans have explicit effects. In the ESB-RS model, plans are written in AgentSpeak for the Jason interpreter. To explain how the action specifications above are used it is necessary to explain the translation of a NoA plan to one in Jason for ESB-RS. The `unstack` plan will be used for an example. This is a plan to move a block from atop another onto the table. In NoA it is as follows:

```
plan unstack (X,Y)
     precondition ( on (X,Y) ),
     effects ( ontable ( X ),
               not on ( X, Y) ),
(
    clear ( Y ) ;
    achieve clear ( X ) ;
    primitive doMove ( X, Table ) ;
)
```

The intuition behind the representation of this plan in ESB-RS is that a plan in NoA is only selected if it is motivated by an obligation and the effects are permitted by the norms according to some deliberation strategy. Jason plans already have a body, signature and precondition as per NoA, so these are the same. The above plan in expressed in AgentSpeak(L) is as follows:

```
+!unstack(X,Y):- block(X) & block(Y) & on(X,Y)
                 & effects(unstack(X,Y))
    <- !achieve(clear(X));
       !move(X,table).
```

Where:

```
effects(unstack(X,Y)) :- effect(unstack(X,Y)) & effect(ontable(X)) &
                         effect(clear(Y) ) & effect(notA( on(X,Y))).
```

The basic principle is that the signature is simply the triggering event, `+!unstack(X,Y)`, the preconditions form the guard and the body is the body of the NoA plan. The effects are added to an `effect(signature)` predicate, in the preconditions. This makes sense, as mentioned above, as a plan can only be chosen if the effects are consistent with the agent's norms and deliberation (it may choose to ignore norms). Each effect is therefore conditioned on the norm structure by the `effect(X)` predicate:

```
effect(X) :-  not ¬allowed(X)
  | (¬allowed(X) & allowed(Y) & moreSpecific(Y,X))
  | (¬allowed(X) & allowed(Z) & sameScope(Z,X) & moreRecent(Z,X))
  | (¬allowed(X) & allowed(W) & intersection(W,X) & moreRecent(W,X)).
```

The above definition is simplified from the actual implementation, which has additional code to handle partial variable instantiations. The action specifications in the expectations as defined in Table 5.4 together with this effect specification predicate effectively implement the plan selection according to the conflict resolution mechanism specified by NoA. This works as follows:

- If the effect (as instantiated) is not explicitly disallowed by a norm, the plan is applicable.
- If this is not the case, it is not applicable, or there is a norm disallowing and another allowing the instantiation – a conflict.
- If it falls into one of the three classes of conflicts, it is allowed if it meets the appropriate resolution strategy:
  - The norm allowing the instantiation of this effect is strictly more specific.
  - The permission has the same scope and is more recent.
  - The scopes intersect and the permission is more recent.

Recency was chosen as the intersection resolution strategy out of those suggested by Kollingbaum [2005] as it was most convenient for implementation. The same general pattern could have been followed and a different function used so like NoA any desired function could be used.

This implementation will find an allowed instantiation of an obliged plan if one exists

that is consistent with the agent's beliefs and the plan's guards – which is exactly what the NoA process achieves. Putting the effects in the plan guard means that Jason automatically attempts to find a valid unification for the variables in the effects and will apply this binding to the other variables in the precondition as appropriate.

The functions above to check the scope (`moreSpecific`, `sameScope`, `intersection`) work by comparing the variables that are unified for the effects parameters. This identifies the various cases as described in the NoA overview above.

Obligations are the motivators for action in NoA and so they must initiate plan selection. Plans are selected in two ways, because they achieve a state obligated by a norm, or because they perform an action obligated by a norm.

Where an obligation specifies "perform(X)", this is handled by intending a plan with that signature. If this fails as the effects of that plan are not true, this means that a norm forbids its execution. This does not mean it should not be carried out, as NoA specifies a norm autonomous agent. In this case the deliberation strategy should be called. NoA does not say how this should be carried out, so the ESB implementation is also minimally prescriptive. It adds a belief that selection failed for that goal and requires expectations conditioned on this to resolve the conflict. Allowing expectations to act on this allows the full ESB machinery to be used for deliberation and whatever social reasoning process is required. The blocksworld example simply relaxes norms until a plan can be intended. This is done by expectations that add a "relaxOC" belief to relax the obligation to clear. As this is a guard on adopting the obligation as a current expectation it effectively removes the norm from the reasoning process. This is only one possible way this could be done and it is an implementation choice. It is not specified by NoA and could have been done in a variety of other ways. It is important to note that the choice of this method to handle deliberation was for speedy implementation rather than a design limit imposed by ESB. There are other possible ways this could have been done. For example, deliberation expectations could themselves cause removal of the norm expectations from the active set through their responses.

Obligations to achieve a state of affairs specify either that a plan to achieve X should be chosen, or that a plan that specifically achieves $\neg$X must be chosen. To implement this selection scheme requires slight extension of the practical reasoner. NoA features a practical reasoner specifically chosen to reason over effects of plans, rather than just preconditions. ESB-RS does not use such a practical reasoner and so the extension is

required. This is not a slight on the power of ESB, as it could have been paired with a different style of reasoner. The extension is implemented as an internal action in Jason to find plans that have X as an effect. This list of plans is then iterated to find one which is applicable given the preconditions and norms held, as described above. If this fails, as per obligations to perform, the deliberation mechanism is used, in the same way.

These sets of expectation structures and support plans described above provide a general way to transform NoA specifications into an ESB representation. This was carried out in ESB-RS and implemented. To further illustrate the implementation a second example more complex than the blocksworld, also from [Kollingbaum, 2005] has been implemented and is described in the following section.

5.2.4 An Extended Example

To demonstrate the NoA to ESB-RS translation works for more complex examples, the Letter of Credit (LoC) protocol example was used [Kollingbaum, 2005, p120]. There are three agents involved here, each with several norms (primarily obligations) which are held and expired as execution progresses. The protocol describes a customer and supplier agent who exchange money for goods using a trusted third party for mediation. This role is played by a bank, who issues a letter of credit to allow the customer to prove to the supplier that it can and will pay, so it is safe to send the ordered goods. Figure 5.6 provides a graphical overview of the process.

A more complex example would be preferable for evaluation purposes, but it is also necessary to ensure that the ESB-RS version behaves as expected compared to the NoA version. As the execution of this protocol is described in detail for NoA it can be seen to be similar to the ESB-RS version.

The key objects reasoned on in NoA are the norms, captured as expectations in ESB, and the effects which are integrated into the plan selection process. This section will step through the key points of the LoC protocol and show how the expectations change according to the norms described in the NoA implementation and the appropriate plan selection.

Figure 5.6 shows the simplified LoC protocol as described by Kollingbaum [2005], with Table 5.5 providing a simplified overview of the plans available to the agents. The

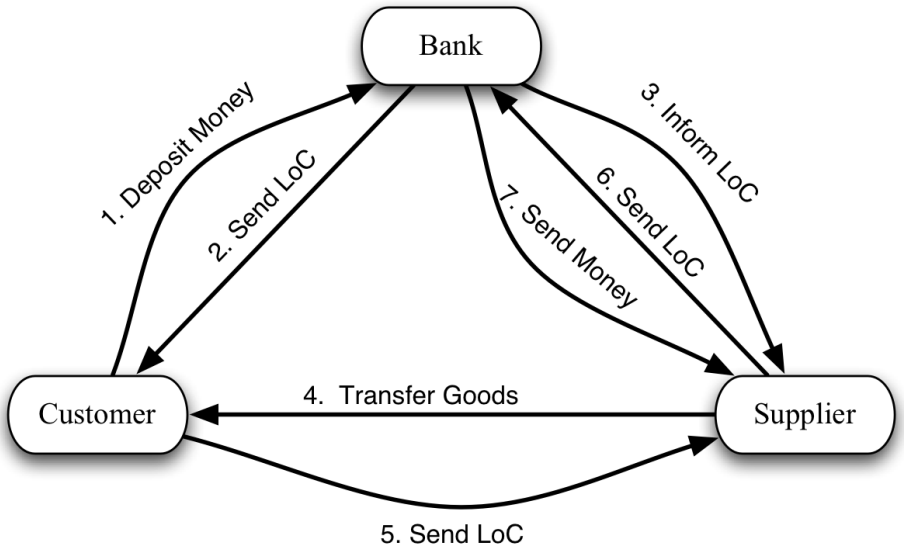

Figure 5.6: The Letter of Credit Protocol (adapted from [Kollingbaum, 2005], p120).

NoA specification has the process captured in the protocol enforced by a set of norms for each party. These are summarised in Figure 5.7, which shows each norm headed with the name of the expectations specifying it. Additionally there is a "prohibRet" norm for all agents that prohibits them from retrieving money from the bank by default.

The following subsections show for each state in the protocol the main changes in the agent's expectations. The format used will be the same as for the soccer example in section 5.1.4.1. As illustrating the complete state for three agents for all 7 protocol states would be overly long and tedious, only the first couple of steps are shown. This is still enough to show the reasoning process in action. The most-relevant expectations will be described in the text and use the names shown in Figure 5.7. The names for the complement expectations are shown in parenthesis.

Table 5.5: Overview of Plans for LoC Example

Customer's Plans

Signature	Preconditions	Effects
depositMoney(Acc,Mon,Bank)	haveAccount(Acc,Bank) haveMoney(Mon)	not haveMoney(Mon) credit(Acc,Mon)
order(Supp,Goods,LoC)	stocked(Supp,Goods) haveLoC(Cust,LoC)	ordered(Cust,Goods)
sendLoC(Supp,Goods,LoC)	haveGoods(Goods) haveLoC(Cust,LoC)	not haveLoC(LoC) haveLoC(Supp,LoC)

Bank's Plans

Signature	Preconditions	Effects
issueLoC(LoC,Acc,Mon,Cust)	credit(Acc,Mon)	haveLoC(Cust,LoC)
payMoney(Supp,LoC,Acc,Mon)	haveLoC(Bank, LoC)	not credit(Acc,Mon) haveMoney(Supp,Mon)

Supplier's Plans

Signature	Preconditions	Effects
transferGoods(Supp,Cust,Goods,LoC)	haveLoC(Cust,LoC) ordered(Cust,Goods)	not stocked(Supp, Goods) haveGoods(Cust,Goods)
retrieveMoney(Supp,Bank,LoC)	haveLoC(Supp,LoC)	not haveLoC(Supp,LoC)

```
Expectation Name (Complement)
Norm Type (
    Role,
  Activity Expression,
  Activation Condition,
  Expiration Condition
)
```

```
obligationDeposit (odComp)
obligation (
  Cust,
  perform
       depositMoney(Cust,Acc,Mon,Bank),
  TRUE,
  credit ( Acc, Mon )
)
```

```
obligationIssueLoC (oilocComp)
obligation (
  Bank,
  perform issueLoC(LoC,Acc,Mon,Cust),
  credit ( Acc, Mon ),
  haveLoC ( Cust, LoC )
)
```

```
obligationOrder (ooComp)
obligation (
  Cust,
  perform order(Cust,Supp,Goods,LoC),
  haveLoC ( Cust, LoC ),
  ordered ( Cust, Goods )
)
```

```
obligationHaveGoods (ohgComp)
obligation (
  Supp,
  achieve haveGoods(Cust,Goods),
  haveLoC(Cust,LoC) and
         ordered(Cust,Goods),
  haveGoods ( Cust, Goods )
)
```

```
obligationSendLoC (oslocComp)
obligation (
  Cust,
   perform sendLoC(Cust,Supp,Goods,LoC),
   haveLoC(Cust, LoC) and
         haveGoods(Cust, Goods),
   haveLoC(Supp, LoC) and
         not haveLoC(Cust, LoC)
)
```

```
permRet (permretComp)
permission (
  Supp,
  perform
      retrieveMoney(Supp,Bank,LoC),
  haveLoC (Supp, LoC),
  haveLoC (Bank, LoC) and
      not haveLoC(Supp,LoC)
      and haveMoney (Supp, Mon)
)
```

```
obligationPay (opayComp)
obligation (
  Bank,
  perform
     payMoney(Bank,Supp,LoC,Acc,Mon),
  haveLoC(Bank,LoC),
  haveMoney(Supp,Mon)
)
```

Figure 5.7: The Letter of Credit norms, presented in protocol order (adapted from [Kollingbaum, 2005]).

The agents are initially in the following states:

Step 1
Customer

EXP_A(**EXP_C**)	Beliefs	Current Plans
ooComp, oslocComp, **prohibRet**, **obligationDeposit**	**¬allowed(retrieveMoney)** perform(depositMoney (Cust,Acc,Mon,Bank))	depositMoney(Acc,Mon,Bank) This plan will deposit money in the bank for a LoC.

Bank

EXP_A(**EXP_C**)	Beliefs	Current Plans
oilocComp, opayComp, **prohibRet**	**¬allowed(retrieveMoney)**	-

Supplier

EXP_A(**EXP_C**)	Beliefs	Current Plans
ohgComp, permretComp, **prohibRet**	**¬allowed(retrieveMoney)** stocked(Goods)	-

As described earlier note how the complement expectations are held (and inactive) for those norms not currently activated. All agents have the expectation *prohibRet* and so believe they are not allowed to retrieve money, this will prevent plan selection that would retrieve money from the bank (unless they were to deliberate otherwise). The customer agent holds the *obligationDeposit* expectation which causes a behaviour to add the perform predicate to trigger plan selection for a specific plan. So this obligation encoded as an expectation motivates the agent to execute the *depositMoney* plan. This represents step one in the protocol as per Figure 5.6. Once this is carried out the agents are then in the states representing step 2:

Step 2

Customer

EXP_A(**EXP_C**)	Beliefs	Current Plans
ooComp, oslocComp, **prohibRet**, odComp	**¬allowed(retrieveMoney)** credit(Acc,Mon)	-

Bank

EXP_A(**EXP_C**)	Beliefs	Current Plans
obligationIssueLoC opayComp, **prohibRet**	**¬allowed(retrieveMoney)** credit(Acc,Mon) perform(issueLoC (LoC,Acc,Mon,Cust))	issueLoC(LoC,Acc,Mon,Cust) This plan will cause the bank to issue the LoC.

Supplier

EXP_A(**EXP_C**)	Beliefs	Current Plans
ohgComp, permretComp, **prohibRet**	**¬allowed(retrieveMoney)** stocked(Goods)	-

As the customer now has credit with the bank, the *obligationDeposit* expectation test became true and as per its response it has removed itself and replaced itself with its complement (which would allow reactivation of the norm in appropriate circumstances). This is the same effect as described by the expiration condition of the norm becoming true. The opposite process has happened with the bank's obligation to issue a LoC. The account credit was instead the activation condition, so the complement expectation *oilocComp* became current with a test that was immediately true and the response applied, which removed itself and added the norm expectation. This will stay current until the LoC is issued at which point it expires.

As it would be overly long to step through the entire protocol, this example shall finish with the example of the suppliers obligation to send the goods which is specified as an achievement rather than a perform activity. This shows the operation of a different part of the norm machinery as implemented in ESB. After the above state, the customer holds an obligation to place an order with the Supplier, which it does. The supplier is informed of the LoC by the bank and is then obligated to ensure the goods are delivered as *obligationHaveGoods* holds. This puts the agents in the states shown below:

Step 4
Customer

EXP_A(**EXP_C**)	Beliefs	Current Plans
ooComp, oslocComp, **prohibRet**, odComp	**¬allowed(retrieveMoney)** haveLoC(Cust,LoC)	-

Bank

EXP_A(**EXP_C**)	Beliefs	Current Plans
oilocComp, opayComp, **prohibRet**	**¬allowed(retrieveMoney)**	-

Supplier

EXP_A(**EXP_C**)	Beliefs	Current Plans
obligationHaveGoods permretComp, **prohibRet**	**¬allowed(retrieveMoney)** stocked(Goods) haveLoC(Cust,LoC) achieve(haveGoods(Cust,Goods))	transferGoods(S,C,G,LoC) The plan loads the goods on a truck to send to the Customer.

Here the supplier agent has *obligationHaveGoods* as a current expectation. This has the action specification to achieve a state of affairs rather than perform a specific action. Therefore the plan is chosen by its effects. The only plan the supplier has to achieve *haveGoods(Cust,Goods)* is *transferGoods*, so this is the plan that gets intended. This shows the ESB version of NoA selecting plans according to effects in contrast to the action based selection in the previous states. The rest of the protocol execution then proceeds in the manner of these steps, with expectations added and removed from the current and active sets to correspond with norm activation and expiration.

5.2.5 Evaluation

The issues with evaluating the quality of the NoA-ESB specification are as with JI. Inspection of execution is harder, as the NoA examples specify only what norms an agent holds and the plans that are executed (compared to the detailed mental states described for JI). The important thing if ESB is claimed to capture NoA reasoning is that the properties that NoA offers are preserved by ESB.

Returning to the key NoA advantages identified in the introduction to this section, it

can be shown they are also present in the ESB-RS design:

Allows for three norm types. The implementation presented here allows specification of obligations, prohibitions and permissions. Their different behaviour is principally captured in the action specifications as per Table 5.4. The blocksworld and letter of credit examples that were implemented illustrate these types of norm in action.

Norms may refer to states or actions. Again the examples require both these types of norm and show that they work. The mechanism to implement this was to distinguish with `perform` and `achieve` predicates in the action specifications. Norms cannot specify conjunctions of states or actions, but this limitation is present in NoA also.

Allows for norm-autonomous behaviour. This is a key principle and requires the agent to be able to deliberate and choose plans inconsistent with its norms. The implemented strategy tries to behave consistently and relaxes norms (based on expectation rules) if it can not. But there is also the case that an agent may wish to be inconsistent even if it need not be. This could be done by changing the deliberation scheme so that it was triggered regardless of plan selection. It would then be able to either intend specific plans, or relax certain norms depending on the deliberation expectations. A limitation would be that to use the result of consistency checking in this deliberation, it would require allowing plan selection to take place and reasoning whether to allow this intention to continue or ending it and re-intending a different plan (to re-trigger selection). Regardless of actual implementation, the point made is that NoA does not constrain how this is done and neither does ESB. Indeed there is greater flexibility offered by the ESB approach.

Detection and resolution of inconsistencies. The requirement for this is that even though inconsistency may be detected the agent can still reason and ignore it. This would be norm autonomous behaviour as per the preceding point. Detection is carried out by capturing the failure to intend a plan and so provides the same information on the source of inconsistency that NoA provides to deliberation. Again it is easy to see how this could be extended in ESB to further enrich deliberation. Possible suggestions could include utilising the knowledge of exactly which norms cause plans to fail.

Detection and resolution of conflicts. Conflicts refer to interactions between permissions and prohibitions and must be resolved. They are dealt with in ESB using the same rules as NoA and this is implemented in the shape of the `effect(X)` predicate and its supporting predicates. From a conceptual point of view this is not ideal – it would be preferable for expectations to handle this, but this is a matter for ESB discussion, in Section 5.4. It could easily be changed to utilise any of the methods mentioned in the NoA thesis.

Although the method of implementation is different, this case study has shown that the key properties of a complex social reasoning system can be preserved when it is specified in ESB. This is important both to evaluate ESB-RS as an implementation, ESB as a theory and therefore the concept of separate social reasoning in general.

Considering the practical social reasoning divide in the NoA implementation, more work is required on the practical reasoner side than would be desirable. This is in part due to the inability to reason over expectation content. Assuming norms are captured with expectations and the conflict resolution function must act on these expectations, detection at least must happen in the BDI level as it cannot be in strategies or behaviours. It is also not surprising that work was required on the practical reasoning side as the NoA scheme is fairly close to practical reasoning in its implementation – it represents an extension to handle external (social) influences.

Not considered in the ESB-RS NoA implementation is the notion of adding new norms previously unknown to the system. This would require an extension to the current ESB implementation to allow for adoption of new expectations. Further work would be required to study the potential pitfalls but it is easy to sketch out how this might be done. When new expectations are added it would be necessary to re-generate the FSM specification for the model checker and then it would be possible to run behaviour-condition style checks on the new expectations to ensure they caused no problems.

A benefit brought to NoA from ESB-RS is that it situates NoA style normative reasoning in an existing agent infrastructure (that of Jason). The NoA architecture presented in [Kollingbaum, 2005] does not situate the reasoner in an agent infrastructure. Combined with the modularity shown in the ESB case studies this presents a strong case for the argument that a framework for general purpose social reasoning can ease implementation of such systems.

5.3 ESB Algorithms for Bounding Reasoning

There are other proposed advantages to generic social reasoning and this section presents some of these. They will be presented in the order they apply in the ESB framework and so firstly the impact of strategies will be considered, followed by general algorithms for behaviours.

5.3.1 Strategies

Strategies in ESB are a means to alter the style of reasoning by controlling which expectations can affect behaviour conditions and how future change in reasoning affects behaviour conditions. As such they have potential to not only alter the behaviour of the agent, but also to bound the reasoning process by reducing the cost to check conditions on the expectation graph.

The strategy in the cards example presented previously in Section 3.2, is a very simple one. It limits graph search by depth, is very simplistic and does not use truly "social" approaches to bounded practical reasoning about the other agent.

ESB could be used to represent much more complex strategies that allow incorporation of distributed, multi-party reasoning. Where nested expectations are allowed, for example, strategies could restrict the depth of nesting, or perhaps even restrict nesting based on some measure of confidence (which could be updated by responses).

Another possibility is to use ideas from decision theory or game theory to influence the design of strategies. For a short example, consider an agent wishing to adopt a minimax strategy for reasoning. The key point of such a strategy is that at every move it is assumed the opponent will use some known evaluation function to pick the worst scoring option for the agent and so for its move the agent will maximise this minimum. To apply this concept to the Rummy example, the "moves" are the observed results of the T tests. The ESB agent's moves are not represented in the expectation graph for simplicity, but its behaviour should maximise this worst-case play by the opponents. The expectation graph is restricted by applying this principle before evaluating the preconditions of behaviours. This requires making assumptions about the evaluation function the opponent uses, which (to keep things simple) might look something like this:

- A pickup scores -1, as they have gained a useful card.
- A discard scores $+1$ as they have lost a card.
- Ignoring a card scores 0 as it tells us nothing.

So the strategy is now to build a tree of the graph to a certain maximum depth (say 2) and then restrict it to only those paths which score lowest (adding the scores down each path to gain a total at the leaves). Assuming state S_4 from Figure 3.2 as initial state, this gives the strategy graph of the shape in Figure 5.8.

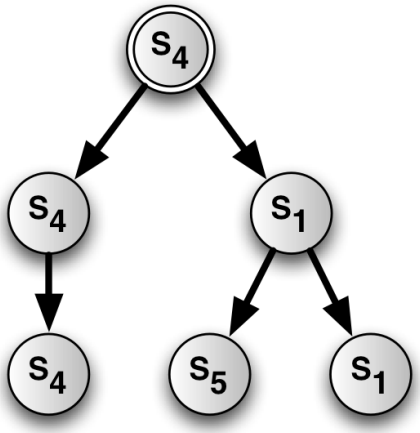

Figure 5.8: Strategy graph based on minimax strategy

This strategy does not include assumptions about the agent's own behaviour. This means that the only influence the mini-max strategy has is to restrict behaviours according to it. If the agent's actions were included in the example, then the strategy graph could be reduced further still by selecting those which maximise the ESB agent's own utility. It does however give a flavour of what more advanced social reasoning strategies might look like and illustrates the power of the framework keeping in mind that we could apply the algorithms described above to such more advanced mutual modelling schemes.

This has only been a brief overview of one possible more advanced strategy to set the scene for further discussion in the next chapter. There is clearly potential in what could be done with strategies and it would form a possible extension to the work here.

5.3.2 Behaviours

The graph-based representation of the reasoning structure provides opportunity for an analysis of the design - particularly to bound reasoning further. This section describes a method that can be used to reduce the number of behaviour rules that need to be checked each reasoning cycle and the complexity of checking them. It is a generic method not dependant on the content of expectations, merely exploiting the structure of ESB. It serves as an example of one way this structure could be exploited for benefits irrespective of the reasoning used.

More specifically, in the Rummy example the behaviour conditions regarding future reasoning are expensive to check, as they involve searching the expectation graph. The algorithm suggested here is to perform offline reasoning at the design stage to simplify run-time reasoning during execution. This off-line reasoning is of course itself computationally expensive, but the argument is that performing this computation once during design is preferable, as it greatly simplifies reasoning during execution.

Behaviours are conditions with actions linked to them, so optimisation takes the form of re-writing the conditions to include state and where possible restricting the conditions yet further to simplify testing. In implementation terms checking for a current state just requires checking the active expectations. The intuitive reasoning behind this algorithm is that considering each behaviour in each state in turn, there are three possibilities:

1. The behaviour condition is true regardless of the combination of expected beliefs that holds. In this case the behaviour can always apply in this state.
2. The behaviour condition is false regardless of the combination of expected beliefs that holds. In this case the behaviour can never apply in this state.
3. Whether or not the behaviour condition applies depends on whether some combination of expected beliefs holds.

The main reason a behaviour condition might always be false (or true) is that it depends on future expectations, which are assumed to be true for the purposes of checking conditions, so only accessibility will matter. In this case it is the state that is important, so including it in the condition can simplify testing. The prospect of dependence solely on state is likely to be common, as it can affect any behaviour rules acting over change in reasoning.

The process of re-writing conditions to include state described above can be carried out with the algorithm shown below. It is possible to have nothing added in the second case, as for every other case the state is considered, so states left out of the condition term will satisfy no other part of the term and will not be considered at all.

INPUT: Accessible expectation graph.
Set of all behaviour conditions $Bconditions = \{B_1, \ldots, B_n\}$
OUTPUT: Set of new behaviour conditions $Bconditions' = \{B'_1, \ldots, B'_n\}$

FOR each condition $B_C \in Bconditions$
create a new condition $B'_C \in Bconditions'$ where $B'_C = false$
END FOR
FOR each $B_C \in Bconditions$
FOR each state $S_i = \{E_1 \ldots E_n\} \in$ *accessible expectation graph*
IF B_C holds for each subset of
$\{\Phi(E_1) \ \ldots \ \Phi(E_n)\}$ **THEN**
Add $\vee \ S_i$ **to** B'_C
ELSE IF B_C does not hold for each subset of
$\{\Phi(E_1) \ \ldots \ \Phi(E_n)\}$ **THEN**
Add nothing
ELSE B_C holds for subset
$C' \subset \{\Phi(E_1) \ldots \Phi(E_n)\} \cup \{\neg\Phi(E_1) \ldots \neg\Phi(E_n)\}$ **THEN**
$\forall \Phi_i \in C'$ **Add** $\vee \ (S_i \wedge \Phi_i)$ **to** B'_C
END IF
END FOR
END FOR

In the previous Rummy example the behaviours were as shown below.

1. (a) Discard 2♢ if $\neg\Phi(E_1) \wedge \neg\Phi(E_2) \wedge \neg\Phi(E_5)$
   (b) Discard 2♣ if $\neg\Phi(E_1)$
2. Discard 3♢ if $\neg\Phi(E_2) \wedge \neg\Phi(E_5)$
3. (a) Discard 2♢ if in all accessible states $\neg\Phi(E_1) \wedge \neg\Phi(E_2) \wedge \neg\Phi(E_5)$
   (b) Discard 2♣ if in all accessible states $\neg\Phi(E_1)$

(c) Discard 3♢ if in all accessible states $\neg\Phi(E_2) \wedge \neg\Phi(E_5)$

Applying the algorithm above to this example results in the simplified set of behaviours shown in Table 5.6.

Table 5.6: New Behaviour Conditions after Expectation Graph Analysis

Name	Action	Original B_C	New B'_C
1(a)	Discard 2♢	$\neg\Phi(E_1) \wedge \neg\Phi(E_2) \wedge \neg\Phi(E_5)$	$(B_C \wedge S_2) \vee$ $(\neg\Phi(E_2) \wedge \neg\Phi(E_5) \wedge S_1) \vee$ $(\neg\Phi(E_1) \wedge S_3) \vee$ $(\neg\Phi(E_2) \wedge \neg\Phi(E_5) \wedge S_4) \vee S_5$
1(b)	Discard 2♣	$\neg\Phi(E_1)$	$S_1 \vee (\neg\Phi(E_1) \wedge S_2) \vee$ $(\neg\Phi(E_1) \wedge S_3 \vee S_4) \vee S_5$
2	Discard 3♢	$\neg\Phi(E_2) \wedge \neg\Phi(E_5)$	$(\neg\Phi(E_2) \wedge \neg\Phi(E_5) \wedge S_1) \vee$ $(\neg\Phi(E_2) \wedge \neg\Phi(E_5) \wedge S_2) \vee$ $(\neg\Phi(E_2) \wedge \neg\Phi(E_5) \wedge S_4) \vee$ $S_3 \vee S_5$
3(a)	Discard 2♢	In all accessible states $\neg\Phi(E_1) \wedge \neg\Phi(E_2) \wedge \neg\Phi(E_5)$	*false*
3(b)	Discard 2♣	In all accessible states $\neg\Phi(E_1)$	$S_3 \vee S_5$
3(c)	Discard 3♢	In all accessible states $\neg\Phi(E_2) \wedge \neg\Phi(E_5)$	S_1

The process does not end here. As some of the behaviours share identical actions (or more accurately prohibitions on discards), they can be combined by only allowing the action when both conditions hold. This operation brings us closer to the original meaning of behaviour rule 3, which was previously expanded to form the three separate sub-rules. This also greatly simplifies the conditions, as for example, forming a conjunction between the conditions of 1(a) and 3(a) in Table 5.6 reduces to *false*. It is easy to build another table that lists each new behaviour against every state as well as the condition required to trigger it. Each entry is either false, or where there is a conjunction of a state and condition, the condition is added to that state and behaviour. Where only a state is present, the condition can be considered to be *true*. For the previous example, this produces Table 5.7 (cells indicate the condition B'_C).

Table 5.7: Reduced State/Behaviour Table

Action / State	Discard 2♢	Discard 2♣	Discard 3♢
1	*false*	*false*	$\neg\Phi(\mathbf{E_2}) \wedge \neg\Phi(\mathbf{E_5})$
2	*false*	*false*	*false*
3	*false*	$\neg\Phi(\mathbf{E_1})$	*false*
4	*false*	*false*	*false*
5	*false*	**true**	*false*

This process infers that we only need to check behaviour conditions in states 1 and 3 and that state 5 always triggers behaviour 1(b). The final simplified behaviours are visualised in a simplified graph shown in Figure 5.9. In this example there is clearly a good reduction in both the number of behaviours checked and the states in which these must be checked. From inspection of Table 5.7 it can be seen that from the original three checks for all five states this design has been reduced to only one check and only in two states.

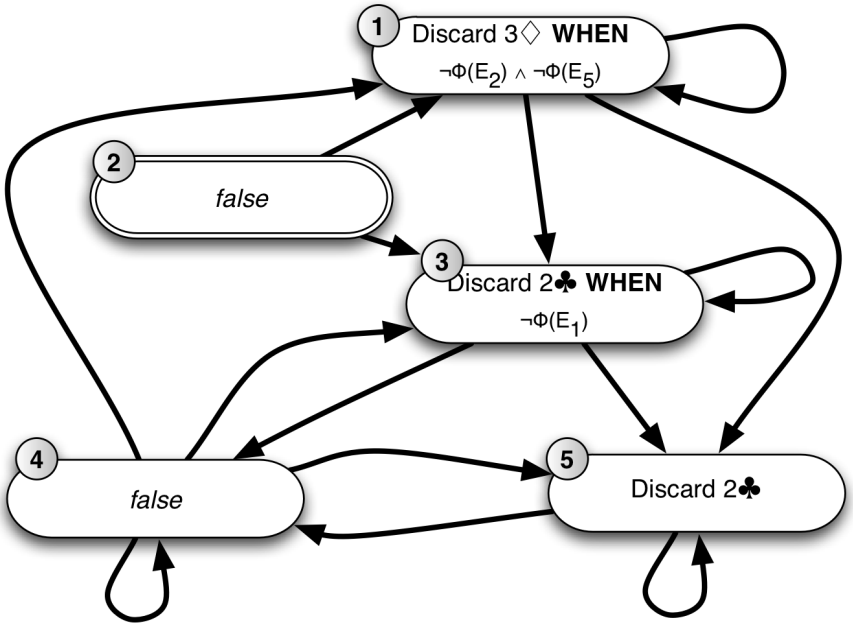

Figure 5.9: Reduced Behaviour Expectation Graph

5.4 Discussion

The starting point for discussion is the hypothesis introduced at the start of this thesis and driving the work described herein:

> It is possible to separate social reasoning from practical reasoning, allowing for a generic specification of social reasoning schemes. This will allow for the development of generic algorithms for bounded execution of agent reasoning and analysis of designs.

The hypothesis is that by using ESB an agent design can take account of typically complex social reasoning problems alongside practical reasoning. This allows the agent's social reasoning to be bounded and parts of the design (and implementation) generalised with automated techniques. Evaluation is a tricky task for this work. The problem tackled represents a new area of work with no other frameworks suitable for direct comparison. Even if direct comparisons could be drawn, there is no obvious choice of metric. Any evaluation must therefore be somewhat subjective and consist of an analysis of the framework with respect to the stated goals presented in Chapter 1.

This can be broken down further into the questions that must be answered:

- What does it mean to separate social reasoning?
- Can social reasoning allow bounded reasoning in general terms?
- When considered separately, how does social reasoning relate to general practical reasoning and what does the interface look like?

The ESB framework addresses these questions with both the theory and ESB-RS implementation. Several case studies have been introduced to demonstrate that social reasoning agents can be created using ESB. The main features that must be evaluated to support the argument for social reasoning are:

- ESB can be used to specify and implement diverse social reasoning schemes.
- Some part of the agent deliberation process for an ESB agent can be automated or carried out algorithmically in a general way.
- The ESB framework allows for general analysis of the design of an agent in ESB, to either study properties or improve an agent implementation by bounding the reasoning.

The criteria for success and failure are hard to define precisely but can be described in general terms. Considering the extremes, success would be that ESB can be shown to capture all possible social reasoning approaches, create implementations of agents based on these and allow placing bounds on their reasoning processes. This is clearly an impractical goal, there is no way to easily define "all social reasoning". Similarly there are a great variety of reasoning processes and possible agent designs, so claims as to the bounding of reasoning would likely be very vague.

The extreme of failure would be the result that it is not possible to represent social reasoning separate from practical reasoning or create a general framework for representation of social reasoning. Bounding reasoning would be found to not be possible, or at least only possible with assumptions of very implementation-specific conditions.

The ESB framework presented here falls in the middle ground, as could be expected, and the work supports the hypothesis. ESB-RS shows that the proposed framework can be implemented, the case studies show that it is possible to represent a variety of social reasoning and examples have been given of how this generic framework might allow for bounding reasoning. Of course assumptions and design decisions have been made that limit certain features or restrict what can be represented. This discussion chapter explores these limits and to what degree ESB supports the original hypothesis through analysis of the case studies and implementation.

The rest of this section studies each of the three main points of the hypothesis above.

5.4.1 Specifying Diverse Social Reasoning Schemes in ESB

One of the principal claims of ESB is that it is sufficiently general to represent a variety of social reasoning schemes. This is central to the claim in the hypothesis. The evidence presented here to support the claim are principally the two case studies of NoA and Joint Intentions. As well as providing examples of different types of reasoning themselves, the process of creating these agents and the resulting designs sheds light on the limitations of ESB.

The case studies were purposefully chosen to be quite different. JI is obviously a good match for ESB as the core principles require an agent to make assumptions about another agent's mental state based on communication. This is conceptually close to ESB with expectations that are conditioned on observations and updated based on more

observation. NoA is quite different and not such an obvious fit. Like most social reasoning it seeks to show how external forces influence an agent's plan selection. In contrast to the mental state based reasoning of JI, these are designed-in norms that the agent holds (or acquires through accepting roles). Chapter 5 showed that both of these approaches can be successfully represented and implemented in ESB.

Creating the designs for the ESB and NoA agents revealed that the generality of ESB can be both a help and a hinderance. There were many different ways that the same reasoning mechanisms could have been expressed in ESB and choosing the best one was not an easy task. NoA with its additional complexity provides the best example of how things perhaps could have been done differently.

Consider the effects that are key to plan selection in NoA. In the presented implementation, they are captured by adding them on to the guards of plans. Another possibility would be to use yet more expectations of the form:

> "When I intend a plan expect its effects."

Aside from the conceptual issues surrounding this – it is clearly not a social reasoning rule – there are other problems. With effects as expectations, norms would have to inspect these expectations.With norms also represented as expectations, plan selection could be handled by behaviours. This would then allow strategies to restrict which norms and effects are considered under certain situations. Without much thought this sounds like a reasonable approach, but it has disadvantages. The main drawback is that unless every effect is considered as a separate case in any conditions, the behaviours (or strategies) would need to reason over expectation content in some way. The formal specification of the framework considers only reasoning about expectations at the meta level, or their direct effects – the currently held Φ beliefs, nested expectations or behaviour actions. The implementation only allows behaviours or expectations to condition on beliefs (added by current expectations or otherwise) and possible explicitly specified future expectations. Support for variables referring to the content of expectations would require a first-order model checker (unless their domains were finite, in which case a transformation would be possible). So with these restrictions on the current versions of ESB and ESB-RS this method of capturing NoA is not feasible.

This is an interesting counter-example to the claim of generality. On one hand, here is a proposed scheme that can not be captured due to current limitations in ESB. On the other, it was possible to implement the same reasoning, albeit a different way. Consider

the worst case, where ESB cannot capture the intended reasoning in any way. ESB-RS considered as a complete system still could provided it was possible within the bounds of the practical reasoner. All the reasoning would be done in the practical reasoner BDI component however. So ESB-RS as an implementation is no less expressive than BDI. Of course this means none of the benefits of ESB could be realised and this would seem to suggest a failure for ESB and the notion of separated social reasoning.

This is unlikely to be the case however. ESB is clearly a very general mechanism especially when thought of as a belief revision mechanism. The conclusion from this analysis is that ESB, and particularly ESB-RS, is a very general framework but care must be taken in agent design to ensure that the best advantages offered are taken. Section 5.4.4.1 offers some insight into designing ESB agents for maximum advantage.

The argument above supports the claim that ESB is general, despite lack of reasoning over content. This is not without cost. The limits in the representation have potential side effects in terms of the number of expectations needed. Where there are a large number of states to reason over and many expectations are required the cost of checking behaviour conditions rises exponentially.

One advantage exhibited well by both examples is how easy extension of a reasoning scheme is with ESB. Both the robosoccer and letter of credit examples required very little extra work to implement. In the case of the letter of credit, this was as the translation from NoA specification was the same and the support plans and behaviours needed were sufficiently general. The robosoccer example better illustrates a desirable property of ESB. As the expectations capture the general property of JI, their modular nature means that additional expectations can be added referring to these to give more functionality to the agents. In this case, to specify when to create JIs, when to accept them and how to carry out a team plan. This modularity is very desirable for the goal of allowing complex implemented social agents, as it promotes code reuse.

The modularity also brings with it the capability to implement social reasoning methods in a somewhat domain independent manner. For both the NoA and JI examples the machinery for social reasoning was implemented using a simple example case initially. The extended examples of robosoccer and the Letter of Credit protocol were created using the existing generic ESB framework for their respective reasoning schemes.

Neither case study required the full machinery of ESB, in particular strategies. This is due to the examples not exhibiting overly complex social reasoning. Many motivating

examples have been given for strategies, for example the minimax approach in the last chapter. Also both approaches used different features of ESB. It is highly likely that if a different example had been chosen strategies may have featured more prominently. The main conclusion that can be drawn from the examples here is that it would require a very complex social reasoning scheme to use all the power that ESB allows, possibly combining multiple existing techniques. This is not unrealistic, to draw a comparison with programming languages, most programs do not use all the features offered. Potential candidates to more fully utilise ESB would, for example, be opponent modelling approaches within some larger agent system. This could not only involve reasoning with nested expectations but also representing system level approaches such as norms and trust within this framework.

5.4.2 General Algorithms for Implementing Social Reasoning

Implementation of social reasoning agents was a big motivation for creating a general framework for social reasoning. Creating the ESB-RS reasoner was important therefore both as a goal in itself and also to allow the case studies to evaluate ESB. ESB-RS exemplifies one possible set of algorithms to process ESB specifications and shows that it is possible to implement generic social reasoning.

At an abstract level the implementation has the same characteristics as the framework more generally, as per the previous section. The main issue from an evaluation standpoint is how closely the implementation matches the ESB theory and by extension how feasible separating social reasoning is in practice.

5.4.2.1 Functionality of ESB-RS Compared to ESB Theory

ESB-RS functionality nearly completely matches that described by the theory. The most obvious omission is any support for nested expectations. They are not included in this first version of ESB-RS but there is no known reason it could not be extended to include them without additional challenges. They were omitted as they were not required for any of the case studies and represent an area for potential future work. They could provide a potentially richer model of JI reasoning, as an agent could have nested expectations capturing the JI directed reasoning scheme of the other agent. This could allow for more advanced commitment strategies as an extension of JI.

Whether nesting is needed at all was a question raised during the background review and when searching for reasoning schemes to use as case studies. There are cases where nested reasoning clearly would be needed, principally bluffing and reasoning about deception. However the current state of MAS research is such that there are few implemented reasoning schemes based on such reasoning due to its complexity. Even if nesting is included in the ESB framework it could be that it does not add much to the expressiveness of the framework and the principal benefit is a more intuitive model. Further work would be required to investigate this at a theoretical level. It may be that it is possible to translate a specification with nested expectations into an similar one where no nesting is present under certain assumptions. A nested expectation graph could be thought of as a "graphs within graphs" structure, and given that both are finite they perhaps transform into a "flat" FSM with no nesting and so a similar structure to non-nested ESB. This is a complex area of further work, as not only would expectations themselves need to be rewritten but also behaviours and strategies conditioned on them.

The inability to reason over expectation content is a limitation deliberately introduced to the framework for simplicity of specification. Further work would be needed to investigate the consequences of removing this limit. It is also a limitation in ESB-RS as a consequence of the abilities of the model checker used. The power of the model checker directly affects what behaviour conditions can be specified and checked, but also brings many potential benefits for the analysis of designs, discussed in the next section.

The ESB framework as a whole lends itself very well to implementation. The graph-like structure to capture the interaction and dynamics of sets of expectations is a most useful model, both for intuitive understanding and creation of algorithms to process it. Being able to use a simple algorithm to transform the ESB specification into a FSM suitable for a model checker was a major benefit for implementation, allowing it to be comparatively simple. Minimal extension to the existing practical reasoner code was required. It can be concluded that the ESB framework is easily implemented without apparently losing significant generality and so general purpose social reasoning as a whole is feasible.

5.4.3 Analysis and Bounding of Agent Designs

In summary, work for this thesis has shown that ESB does allow for a generic framework for the analysis of agent designs and bounding reasoning, but further work would be required to create case studies and evaluate the practicalities of this further.

Section 5.3 presented theoretically two possible ways in which agent reasoning could be bounded, beyond the trivial example strategies shown earlier. The strategy framework in ESB provides a very general purpose mechanism for an agent designer to constrain the social reasoning of an agent. In broad terms this supports the argument that bounded reasoning principles would be a benefit of generic social reasoning. The effectiveness of the methods to bound reasoning is a matter of importance to evaluate the extent to which this is a benefit of ESB. Effectiveness is likely to be very domain specific, as a strategy that is suitable for one domain may have very different effects on another. So whilst the strategy mechanism is general, specific strategies may not be.

The manner in which ESB bounds reasoning can be compared with BDI – as a common generic agent reasoning architecture it is a natural comparison. BDI bounds by effectively restricting planning to the immediate future and revising plans and intentions based on new information. With strategies, ESB bounds by limiting the scope of future reasoning states that are considered and it is these that change based on observation.

The algorithm described to rewrite behaviour conditions to include state is domain-independent, although its effectiveness will depend on how correlated expectation states and behaviour actions are. Behaviour conditions are only simplified by the example algorithm where they can be replaced with simple checks on the expectation graph state (the content of EXP_A). It is easy to see that in many cases one would see a strong correlation. For example, in a normative system such as the NoA case study, agents may have their actions constrained by norms and therefore behaviour conditions closely related to the expectation state (as defined by held norms). In general, it is not unusual for agents to act in a certain limited way according to their situation. ESB provides an excellent way in which to consider only the reasoning rules appropriate to the current situation.

The algorithm to rewrite states is complex however, as each condition must be checked for each combination of expectations. It is possible this could cause problems in very large implementations, but this seems unlikely as the checks to be made are simple

and the processing is not required at runtime (indeed its purpose is to reduce runtime computation). The checks are simpler than runtime behaviour checks, as they need only check if an expectation includes a term affecting the behaviour. Similarly although each resulting behaviour's action would need compared against every other to combine them further as suggested, the test is a simple one of equality.

Rewriting behaviour conditions is only one possible refinement of the ESB algorithm to reduce run-time computational effort with specification pre-processing. This represents another area of possible further study. As well as additional algorithms, this could include general statistical analyses of these, perhaps using generated expectation graphs to explore the types of structure that were most effectively reduced.

Another promising area for future investigation comes from the specifics of the ESB-RS implementation, rather than the general properties of the theory. As ESB specifications are transformed for use with a model checker, it would be possible to use it to check properties of the design. Common uses of model checkers include checking race conditions and deadlock and these properties could be relevant for MAS design. An analogy for race conditions could be a design that depends on two states for an agent being mutually exclusive – like trusting or distrusting a seller – and the designer would want to ensure that this is in fact the case. Deadlock detection could be relevant where expectations capture states in protocols, for example.

5.4.4 Discussion of the ESB Implementation

This section is concerned with issues raised during the implementation that pertain to the effectiveness of ESB, and separate social reasoning in general, that do not fit the previous discussion.

The two biggest design decisions for ESB-RS that affect the types of systems it can specify are the handling of responses and the separation of $T+$ and $T-$ tests. The mechanism to handle the order in which tests are applied was thoroughly discussed where it was introduced in Section 4.2.3. The main reason to split the test into two cases, rather than either T or $\neg T$, is to allow a greater variety of test conditions. In particular, this allows for easy implementation of timeout cases. Either an event is observed (T) or a timeout reached ($T-$) and then the expectations updated. In general, implementation choices were those giving the greatest freedom for agent specification.

5.4.4.1 Designing ESB agents

Finally, some discussion on the process of creating ESB agents seems appropriate. As a framework for agent reasoning ESB not only places constraints on the properties of the agents that may be specified with it but also on the process of designing agents. The goal here is not to present an agent design methodology – that is another entirely different field of agent-orientated programming research, closely allied to software engineering. The purpose of this section is to provide some observations gleaned through the implementation process.

As Section 5.4.1 discusses, there were different ways the NoA plan effects could have been coded. The key to creating the implementation was to focus on the outcome of the reasoning process and work backwards from there. This is similar to coding BDI agents, which is done bottom-up from the plans, compared to the theory which describes reasoning from the top down, starting with desires. The approach taken to design the normative agents in ESB was to consider first what factors that plan selection depended on, then how the expectations could influence these.

This is directly related to the relatively low level focus of the NoA approach being implemented however. For JI which is a more abstract concept, the design approach was slightly different. Here the approach focussed on the core concept of ESB – inferring beliefs required for decisions based on observed events. There is a very natural match here as the notion of commitment requires that an agent makes an assumption about the other's mental state based on communication.

The choice of which reasoning to encode in the expectations (rather than behaviours, strategies or plans) brings with it another debate, that of "ESB abuse". As the expectation mechanism is so general, it is easy to fall into the trap of putting practical reasoning into expectations, or processes that are not strictly observations linked to inferred reasoning. Whilst there is no restriction to prevent this, it should be cautioned against. A great deal of the modularity of the ESB approach comes from the separation of the reasoning. To enable reuse and comparison of agent designs there should be some uniformity. ESB aims to impose this and provides corresponding benefits, as discussed throughout this chapter.

Finally care must be taken in designs to work around the limitations of reasoning over expectation content. Whilst it is fine for a behaviour to condition on a specific future expectation, it is not possible to condition on a property of future expectations content

in general. Theoretically this is not a problem where the expectations are bounded and fixed, but from a programming standpoint it would be convenient to use variables in both expectations and behaviours. This is possible to a limited extent, as variables may be used in expectations and in the part of behaviours that checks the agent's belief base. Content of expectations can effectively be reasoned on where it is relating to current expectations, as their expected Φ values will be in the belief base. In practice for the case studies performed this was not an overly burdening restriction. This is the most significant limitation in ESB however. It makes ESB specifications more verbose and harder to write.

5.5 Summary

The aim of this chapter was a critical discussion of the merits of general separate social reasoning, as realised through the ESB theory.

The bulk of the chapter is formed by the two case studies, which take the form of existing social reasoning schemes specified in ESB and implemented in ESB-RS. These serve to demonstrate that ESB can capture different kinds of reasoning and that ESB-RS can be used to create ESB agents and execute these specifications.

The section on bounding agent reasoning forms a more conceptual study of ESB. As a proposed benefit of general social reasoning schemes is to allow for general purpose algorithms for their simplification or analysis, these provide examples of how this might be done. There is no implementation for these examples, so there are no empirical results for consideration. It is doubtful that much could be shown - there is no such thing as "typical" agent social reasoning and so statistics on the reduction of states or cost of reasoning are likely very domain dependant. They do however show that such schemes are possible to create.

The final section discussed issues arising from the process of creating the case studies and comparison between them. This was done with reference back to the hypothesis in Chapter 1 and supported the three main claims made with examples from the case studies.

Chapter 6

Conclusions

This thesis has investigated the idea of specifying and implementing agent social reasoning as a separate component to practical reasoning. The motivation was that a generic model of social reasoning might bring similar benefits to MAS that approaches such as BDI have in the practical reasoning realm. This goal was borne from several observations of MAS in general:

- A lot of social reasoning is done by the designer, rather than the agent.
- Many approaches focus only on a single facet of social reasoning rather than the general case.
- There may be common structure in social reasoning that could be exploited, by way of a general framework, to bound agent reasoning, compare and analyse designs.

The Expectation-Strategy-Behaviour (ESB) framework has been presented as one possible way in which generic social reasoning could be captured. The ESB framework has been described not only at a theoretical level, but also in terms of algorithms to implement a reasoner based on it. The ESB-RS implementation has also been developed and several case studies of existing social reasoning schemes implemented in ESB-RS were studied. These studies were then used as a focal point for discussion of the merits of the ESB approach and generic social reasoning in general.

This section will present a summary of the key points made in each chapter, followed by an evaluation of the main contributions of this thesis. Finally possible future work will be recapped and a few concluding remarks made.

6.1 Summary

The background chapter began with an overview of agent reasoning in general, defining the differences between what is classically thought of as practical reasoning and the approaches referred to in this thesis as social reasoning. The bulk of the chapter relates to existing work that motivates and is relevant to the goals of this thesis. Examples of many types of reasoning were given, grouped roughly into areas based on how the reasoning is controlled. After a summary of the main motivating conclusions drawn from this study, as repeated above in the introduction, the chapter moves onto work related to the approach taken in this thesis. This shows that whilst there is research that shares either some of the motivation or is in some way similar in execution, there is little that could be considered similar enough for direct comparison. This in itself encourages the novel approach taken in this thesis.

Chapter 3 presented the theoretical formal overview of the ESB theory. This starts with an illustrative Rummy cards example and proceeds with the logical definition using $\mathcal{LORA}$. The important concepts of expectations, strategies, behaviours are defined, along with the key intuition of the expectation graph. The chapter finishes with the theoretical extension to the formalism to allow for nested expectations. Although this has only been taken as far as the formalism, nesting of expectations represents an important class of complex reasoning. Much desirable behaviour for agents requires reasoning about another's reasoning process. Nesting would be required to capture this elegantly in ESB.

The description of the ESB-RS system is introduced next. This is split into two parts, firstly the algorithms that are required, followed by the details of the implementation. This chapter describes the Jason BDI reasoner, which was extended to create ESB-RS. An important innovation for the ESB-RS implementation is the use of the model checker NuSMV to build a representation of the expectation graph and allow efficient checking of behaviour conditions. As well as easing the implementation this brings significant benefits in the analysis of agent designs, allowing static checking of expectation properties.

The final chapter represents the evaluation component of the thesis. As there is no directly comparable framework to evaluate ESB against, the evaluation must be more subjective. The chosen method was to illustrate the generality of ESB by implementing two case studies of existing social reasoning schemes, Joint Intentions and the NoA

model of normative reasoning. Analysis and discussion of these case studies and the process of creating the designs allows conclusions to be drawn regarding the effectiveness and benefits of ESB. This in turn reflects on the hypothesis presented here, supporting the claims of the utility of general purpose social reasoning. As well as the implemented case studies, a couple of algorithms for bounded reasoning are also presented. These utilise the structure imposed by ESB on a specification to realise improvements in the cost of checking behaviour conditions, or in the case of the strategy, present a way to control the agent's reasoning.

6.2 Summary of Contributions

The main contributions of this thesis are as follows:

- A theoretical framework for general social reasoning.

  An important contribution of this thesis is the ESB framework for social reasoning. It shows that it is possible to create a general reasoning framework specifically for social reasoning. This is important, as detailed specification is necessary for discussion and allowed creation of an implementation.

- An implementation allowing the execution of concise and modular declarative social reasoning rules.

  The ESB-RS implementation of an ESB reasoner is the main software artefact produced by the work developing this thesis. It shows that it is possible to implement ESB and more generally that separate social reasoning can bring benefits to agent implementation. It represents a framework generic enough to capture a variety of reasoning methods but with enough structure to develop general methods to bound reasoning and analyse agent designs. A closely related contribution is the discussion of how an agent designer might go about using this to create agents.

- Implementations of JI and NoA in ESB-RS.

  As well as allowing evaluation the ESB-RS framework, the implemented examples also helped to drive development by providing concrete problems to solve. Additionally they serve to illustrate the working of ESB in realistic usage scenarios.

- The results show that social reasoning can be considered separately to practical reasoning.

  This is the key result supporting the hypothesis. Whilst the ESB theory and implementation provide one possible way to create a social reasoner, the wider implication is that there is practical use to this as a concept.

- A framework for bounded social rationality using generic algorithms or methods.

  The ESB-RS implementation shows that it is possible to bound agent reasoning in a practical way. This allows a designer to place limits on the scope of an agent's social reasoning process.

- A framework to allow separation of social and practical reasoning concerns.

  As well as showing that social reasoning can be separated from practical reasoning, the ESB framework also provides a means to implement this separation. This logical separation is an advantage in terms of modularity of agent designs and in comparison of social reasoning approaches.

- Discussion of the merits of general social reasoning, distinct from practical reasoning.

  To take a more general view, this thesis creates a discussion of the merits of general social reasoning as an alternative approach to purely practical reasoning. The overall message is that there may be benefits in a less general agent framework, as social reasoning can benefit from specific attention.

6.3 Future Work

Despite the many contributions of this thesis there are many possible areas of future work. They have been highlighted mainly through limitations met in the case studies and discussed in the later chapters. The approach taken during the development of this thesis made it inevitable that there would be lots of potential to extend this work. The main output here is the ESB model and implementation, created to show that general purpose social reasoning is a viable concept. The focus throughout was getting a complete theory and implementation, to enable the creation of example agent systems. As such the focus was getting each part "good enough" rather than developing a particular

component in depth. Therefore many areas could warrant future research:

- Extension of the ESB-RS implementation to include nesting.

  The ESB-RS implementation does not currently handle Φ components that are themselves expectations. The theoretical discussion of nesting describes how the extension could look and the various expectation sets processed. This provides an insight into what this extension might look at, but there are undoubtably problems faced requiring further work. Implementation of nesting would also raise the theoretical question of if nesting is required at all, or is it just a useful design intuition? Investigation into this would involve consideration of the nested and non-nested expectation graph structures and if a transformation is possible in the general case.

- Static analysis of agent designs (via the model checker or otherwise).

  One of the biggest potential benefits of a generic framework for social reasoning is the support it could provide both to agent designers and researchers investigating the properties of existing agent reasoning schemes. The simple examples given in this document represent only a cursory consideration of what might be possible.

- More complex strategies.

  Another area that shows much promise is that of strategies, in the ESB sense. They appear to offer great potential for both implementing new types of reasoning and bounding existing reasoning in useful ways. The examples used in this thesis have not required complex strategies, but now ESB is more fully developed this could allow their creation.

- More examples and combinations of different reasoning schemes.

  One of the great strengths revealed by the example-led evaluation of ESB-RS was how the modular nature of ESB specifications lend themselves well to extension. This suggests opportunities to combine existing reasoning approaches in novel ways.

- Reasoning over expectation content.

  The main limitation of ESB theory (and implementation) discovered during evaluation was the inability to reason over the content of expectations. Extension of

ESB to allow references in strategies and behaviours to expectations not by name but by matching conditions on their content could bring many benefits. Principally this would be a benefit for agent implementation, allowing simpler more concise strategy specification. The hurdle to implementation of such a scheme would be the current method's dependance on a model checker for behaviour condition checks. This would require a first-order model checker or perhaps a theorem prover base approach.

- Adoption of new expectations.

  The current framework assumes that the set of all possible expectations is finite and known at design time. From an implementation perspective the ability for an agent to add new expectations at runtime and regenerate the expectation FSM seems straightforward. However this would require further formal study to investigate the implications.

- Tools to create ESB agents.

  The potential for static analysis of agent systems using ESB presents an opportunity for tools to aid the design of ESB agents. These could perhaps take the form of graphical tools to visualise expectation graphs, or a set of standard tests to identify weaknesses in agent designs.

6.4 Concluding Remarks

This thesis has presented the ESB framework for bounded agent social reasoning as one possible means to address the concept of representing social reasoning separately from practical reasoning. A complete implementation of the theory was presented in ESB-RS, which allows an agent designer to implement abstract social reasoning schemes simply by specifying their properties and a structured way. Execution of this specification is carried out by ESB-RS, which provides many opportunities for checking properties of the design, bounding reasoning and easily extending the agent's capabilities.

Appendix A

Technical Details of ESB-RS Implementation

This appendix contains further technical details on the ESB implementation that were not appropriate for the main text. A complete copy of all ESB-RS code can be found on the CD submitted with this thesis, or in the Agents Group CVS repository in the School of Informatics. Access is available on application, please see `http://www.cisa.inf.ed.ac.uk/agents/` for contact details.

A.1 XML Schema

A.1.1 ExpectationSet.dtd

```
<!ELEMENT ExpectationSet (expectation*)>
<!ELEMENT expectation (agent, name, condition, phi, test_plus,
   test_minus, rho_plus, rho_minus)>

<!--name of the agent holding this exp-->
<!--
This should be the name of an agent in the system. "self"
should evaluate to the name of the agent. Nesting would be
the only reason this should be anything else.
-->
<!ELEMENT agent (#PCDATA)  >
```

```
<!-- name of the expectation -->
<!--
This is just a string. The only context it should be used
in is that of matching from the various response sets.
-->
<!ELEMENT name (#PCDATA)>

<!--the condition under which this exp applies-->
<!--
Conditions and Tests are effectively the same. Both
must evaluate to T or F, based on the current
belief base.

These are defined the same as guards on Jason plans.
-->
<!ELEMENT condition (#PCDATA)>

<!--the expected belief-->
<!--
This should just be a straight belief. It will be
added to the BDI belief base. Basically if C holds,
it should be the case that the agent believes phi.
-->
<!ELEMENT phi (#PCDATA)>

<!--the T test-->
<!--
See the condition.
-->
<!ELEMENT test_plus (#PCDATA)>
<!ELEMENT test_minus (#PCDATA)>

<!-- The responses. Each defined in terms of
exps to add, and remove -->
<!ELEMENT rho_plus (addSet, remSet)>
<!ELEMENT rho_minus (addSet, remSet)>
```

```
<!ELEMENT addSet (name*)>
<!ELEMENT remSet (name*)>
```

A.1.2 Strategy.dtd

```
<!--Dead simple, just two TRANS statements  -->
<!ELEMENT strategy (main,expectation)>

<!--These will be TRANS statements in the main FSM  -->
<!ELEMENT main (transStatement*)>

<!--These will be TRANS statements in each expectation FSM  --
   >
<!ELEMENT expectation (transStatement*)>

<!--A valid NuSMV transition constraint statement -->
<!ELEMENT transStatement (#PCDATA)>
```

A.1.3 Behaviours.dtd

```
<!ELEMENT BehaviourSet (behaviour*)>

<!ELEMENT behaviour (condition,action)>

<!ATTLIST behaviour name CDATA #REQUIRED>

<!--
 a condition is a conjunction of a NuSMV CTL formulae
 and a jason guard.
  jason_guard can be anything you can specify in the
  guard of a jason plan this gives us negation, so
  with the AND we have a propositional language.

 NOTE part of the reason this is designed as it is,
 with both parts mandatory is to make for easy parsing.
```

```
-->
<!ELEMENT condition (ctl,jason)>
<!ELEMENT ctl (#PCDATA)>
<!ELEMENT jason (#PCDATA)>

<!--An action is a belief to add to the jason belief base -->
<!ELEMENT action (#PCDATA)>
```

A.1.4 esb-template.smv

This listing is for the template file used to generate the FSM. This is read in by the Java code that parses the expectation specifications, then written out with the appropriate expectation macros prepended to it. The model checker then runs it though the CPP pre-processor before execution to expand the macros into a complete SMV specification.

```
--this is a template to generate an E-graph spec
#define expInstance(NAME) NAME : expectation({True,False,DC},
   NAME,self);
#define addStatement(NAME,OTHER,CONDITION) (NAME.ExpC != True)
    & (next(OTHER.Test) = CONDITION) :False;
#define remStatement(OTHER,CONDITION) (next(OTHER.Test) =
   CONDITION) : DC;

#define expList {none,EXPLIST}

MODULE main

VAR
  EXPINSTANCES

--this is the global test
  Test : {Tp,Tm,NA};
  TestAppliesTo : expList;
```

```
ASSIGN

  init(Test) := NA;
  init(TestAppliesTo) := none;
  next(Test) := {Tp,Tm,NA};
  next(TestAppliesTo) := expList;

  EXPCSTATEMENTS

  SPECSTOCHECK
#ifdef STATE
INIT
  STATE
#endif

#ifdef MAINSTRAT
  MAINSTRAT
#endif

MODULE expectation(initial,name,main)
VAR
  ExpC : {True,False,DC};
  Test : {Tp,Tm,NA};
ASSIGN
  init(ExpC) := initial;

  init(Test) := NA;
  next(Test) := case
                 (next(main.TestAppliesTo) = name) & (ExpC =
                    True) : next(main.Test);
                 (Test = Tp) | (Test = Tm) : NA;
                 1 : Test;
                esac;

DEFINE
  phi := case
           ExpC = True  : True;
```

```
            ExpC = False : False;
            ExpC = DC : False;
          esac;

  ExpCUpdate := case
                  ExpC = True : {True,False};
                  ExpC = False : {True,False};
                  ExpC = DC : DC;
                esac;
#ifdef EXPSTRAT
  EXPSTRAT
#endif
```

Bibliography

AgentLink. AgentLink case studies. URL `http://www.agentlink.org/resources/casestudies.html`. Last checked 26/8/2010.

Alexandros Belesiotis, Michael Rovatsos, and Iyad Rahwan. Agreeing on plans through iterated disputes. In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, 2010.

Guido Boella and Leendert van der Torre. Normative multiagent systems and trust dynamics. *Trusting Agents for trusting Electronic Societies, Lecture Notes in Artifical Intelligence*, 3577:1–17, 2005.

Guido Boella and Leendert van der Torre. Norm negotiation in multiagent systens. *International Journal of Cooperative Information Systems (IJCIS) Special Issue: Emergent Agent Societies*, 16(1):97–122, Mar 2007.

Guido Boella, Joris Hulstijn, and Leendert Van Der Torre. Virtual organizations as normative multiagent systems. *Proceedings of the Annual Hawaii International Conference on System Sciences*, page 192, 2005. ISSN 1530-1605.

Rafael H. Bordini and Jomi Fred Hübner. Jason: A java agentspeak interpreter, 2009. URL `http://jason.sourceforge.net`. Last checked 18/12/2009.

Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007.

Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Comput. Intell. (Canada)*, 4(4):349 – 55, 1988. ISSN 0824-7935.

Cosmin Carabelea and Olivier Boissier. Coordinating agents in organizations using

social commitments. *Electronic Notes in Theoretical Computer Science*, 150(3):73 – 91, 2006. ISSN 1571-0661.

David Carmel and Shaul Markovitch. Learning models of intelligent agents. In *Proceedings of the National Conference on Artificial Intelligence*, volume 1, pages 62 – 67, 1996.

Cristiano Castelfranchi, Frank Dignum, Catholijn Jonker, and Jan Treur. Deliberate normative agents: Principles and Architecture. *Lecture Notes in Artificial Intelligence*, 1757:364–378, 2000.

Lawrence Cavedon and Gil Tidhar. A logical framework for multi-agent systems and joint attitudes. In *Proceedings of First Australian Workshop on DAI, Distributed Artificial Intelligence. Architecture and Modelling.*, pages 16 – 30, 1996.

Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499. Springer, July 1999.

Phillip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artif. Intell. (Netherlands)*, 42(2-3):213 – 61, 1990. ISSN 0004-3702.

Phillip R. Cohen and Hector J. Levesque. Teamwork. *Nous*, 25(4), 1991.

Phillip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Lecture Notes in Artificial Intelligence*, 2650:1 – 36, 2003.

R. Conte, C. Castelfranchi, and F. Dignum. Autonomous norm acceptance. In *Intelligent Agents V: Agent Theories, Architectures, and Languages, Proceedings of the 5th International Workshop, ATAL'98*, pages 99 – 112, 1999.

Stephen Cranefield. A rule language for modelling and monitoring social expectations in multi-agent systems. In *Lecture Notes in Computer Science*, volume 3913, pages 246 – 258, 2006.

Stephen Cranefield. Modelling and monitoring social expectations in multi-agent systems. In *Lecture Notes in Computer Science*, volume 4386, pages 308 – 321, 2007.

Keith Decker and Victor Lesser. Designing a Family of Coordination Algorithms.

Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 73–80, 1995.

Robert Demolombe and Vincent Louis. Speech acts with institutional effects in agent societies. In *Deontic Logic and Artificial Normative Systems, Proceedings of the 8th International Workshop on Deontic Logic in Computer Science, DEON 2006. (Lecture Notes in Computer Science Vol.4048)*, pages 101 – 14, 2006.

Louise A. Dennis, Berndt Farwer, Rafael H. Bordini, Michael Fisher, and Michael Wooldridge. A common semantic basis for BDI languages. In *Lecture Notes in Computer Science*, volume 4908, pages 124 – 139, 2008.

Frank Dignum, David Morley, Elizabeth A. Sonenberg, and Lawrence Cavedon. Towards socially sophisticated BDI agents. In *Proceedings of the Fourth International Conference on MultiAgent Systems*, pages 111 – 18, 2000.

Frank Dignum, David Kinny, and Liz Sonenberg. Motivational attitudes of agents: on desires, obligations, and norms. In *Lecture Notes in Artificial Intelligence*, volume 2296, pages 83 – 92, 2002a.

Frank Dignum, David Kinny, and Liz Sonenberg. From Desires, Obligations and Norms to Goals. *Cognitive Science Quarterly*, 2(3-4):407–430, 2002b.

Marc Esteva, Juan A. Rodriguez-Aguilar, Josep Ll. Arcos, Carles Sierra, and Pere Garcia. Institutionalising open multi-agent systems. In *Proceedings Fourth International Conference on MultiAgent Systems*, pages 381 – 2, 2000.

Dov M. Gabbay, Mark A. Reynolds, and Marcello Finger. *Temporal Logic: Mathematical Foundations and Computational Aspects*, volume 2. Oxford Science Publications, 2000.

Michael Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings of the sixth national conference on Artificial Intelligence (AAAI-87)*, pages 677–682, 1987.

Michael Georgeff, Barney Pell, Martha Pollack, Millind Tambe, and Michael Wooldridge. The belief-desire-intention model of agency. *Lecture Notes in Artificial Intelligence*, 1365, 1999.

Davide Grossi, Frank Dignum, Mehdi Dastani, and Lamber Royakkers. Foundations of

organizational structures in multiagent systems. In *Proceedings of the International Conference on Autonomous Agents*, pages 827 – 834, 2005.

IFAAMAS. Proceedings of the eighth international conference on autonomous agents and multiagent systems, 2009. URL `http://ifaamas.org/Proceedings/aamas09/index.html`. Last checked 18/04/2010.

Nick R. Jennings. Commitments and conventions: the foundation of coordination in multi-agent systems. *Knowledge Engineering Review*, 8(3):223 – 50, 1993. ISSN 0269-8889.

Shakil M. Khan and Yves Lesperance. ECASL: A model of rational agency for communicating agents. In *Proceedings of the International Conference on Autonomous Agents*, pages 891 – 898, 2005.

Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems*, 2008.

Martin Kollingbaum. *Norm-governed Practical Reasoning Agent*. PhD thesis, University of Aberdeen, 2005.

Mamadou T. Kone, Akira Shimazu, and Tatsuo Nakajima. The state of the art in agent communication languages. *Knowledge and Information Systems*, 2(3):259 – 84, 2000. ISSN 0219-1377.

Sanjeev Kumar, Marcus J. Huber, Philip R. Cohen, and David R. McGee. Toward a formalism for conversation protocols using joint intention theory. *Computational Intelligence*, 18(2):174 – 228, 2002. ISSN 08247935.

Rodrigo Machado and Rafael Bordini. Running agentspeak(l) agents on sim-agent. In *Lecture Notes in Artificial Intelligence*, volume 2333, pages 158 – 74, 2002.

Jarred McGinnis. *On the Mutability of Protocols*. PhD thesis, Centre for Intelligent Systems and Applications, School of Informatics, University of Edinburgh, 2006.

Khaled Nagi. *Overview of Agent Technology*, volume 2249 of *Lecture Notes in Computer Science*, chapter 3, pages 12–39. Springer-Verlag, 2001.

NASA. Remote agent website. URL `http://ti.arc.nasa.gov/tech/asr/planning-and-scheduling/remote-agent/`. Last checked 26/8/2010.

Matthias Nickles, Michael Rovatsos, Wilfried Brauer, and Gerhard Weiß. Towards a Unified Model of Sociality in Multiagent Systems. *International Journal of Computer & Information Science*, 5(2), 2004a.

Matthias Nickles, Michael Rovatsos, and Gerhard Weiß. Formulating agent communication semantics and pragmatics as behavioral expectations. *Agent Communication. International Workshop on Agent Communication, AC 2004. Revised Selected and Invited Papers (Lecture Notes in Artificial Intelligence Vol. 3396)*, pages 153 – 72, 2004b.

Nardine Osman and David Robertson. Dynamic verification of trust in distributed open systems. In *International Joint Conference in AI*, 2007.

Jigar Patel, W.T. Luke Teacy, Nicholas R. Jennings, Michael Luck, Stuart Chalmers, Nir Oren, Timothy J. Norman, Alun Preece, Peter M. D. Gray, Gareth Shercliff, Patrick J. Stockreisser, Jianhua Shao, W. Alex Gray, Nick J. Fiddian, and Simon Thompson. Agent-based virtual organisations for the grid. In *Proceedings of the International Conference on Autonomous Agents*, pages 1261 – 1262, 2005.

Iyad Rahwan, Sarvapali D. Ramchurn, Nick R. Jennings, Peter McBurney, Simon Parsons, and Liz Sonenberg. Argumentation-based negotiation. *Knowledge Engineering Review*, 18(4):343 – 75, 2003. ISSN 0269-8889.

Sarvapali D. Ramchurn, Dong Huynh, and Nicholas R. Jennings. Trust in multi-agent systems. *Knowledge Engineering Review*, 19(1):1–25, 2004.

Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. *Agents Breaking Away. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '96 Proceedings*, pages 42 – 55, 1996.

Michael Rovatsos. Dynamic semantics for agent communication languages. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007.

Rummy.com. The Rules of Rummy, 2008. URL `http://rummy.com/rummyrules.html`. last checked 29/09/2008.

Sabyasachi Saha, Anish Biswas, and Sandip Sen. Modeling opponent decision in repeated one-shot negotiations. In *Proceedings of the International Conference on Autonomous Agents*, pages 535 – 541, 2005.

Paul Scerri, Tracy Von Gonten, Gerald Fudge, Sean Owens, and Katia Sycara. Transitioning multiagent technology to UAV applications. In *Proceedings of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, 2008.

John R. Searle. *Speech Acts: an Essay in the Philosphy of Language*. Cambridge University Press, 1969.

Michal P. Sindlar, Mehdi M. Dastani, Frank Dignum, and John-Jules CH. Meyer. Mental state abduction of BDI-based agents. *Lecture Notes in Computer Science*, 5397: 161–178, 2009.

Ira A. Smith, Phillip R. Cohen, Jeffery M. Bradshaw, Mark Greaves, and Heather Holmback. Designing conversation policies using joint intention theory. In *Proceedings International Conference on Multi Agent Systems*, pages 269 – 76, 1998.

University of Michigan Soar Group. Soar home page, 2009. URL `http://sitemaker.umich.edu/soar/home`. Last checked 18/12/2009.

Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83 – 124, 1997. ISSN 1076-9757.

Bob van der Vecht, Frank Dignum, and John-Jules. Ch. Meyer. Autonomy and coordination: Controlling external influences on decision making. In *2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 92 –95, sept. 2009.

H. Van Dyke Parunak, Sven Brueckner, Mitch Fleischer, and James Odell. A design taxonomy of multi-agent interactions. *Agent-Oriented Software Engineering IV. 4th International Workshop, AOSE 2003. Revised Papers. (Lecture Notes in Computer Science Vol.2935)*, pages 123 – 37, 2003.

Jose M. Vidal and Edmund H. Durfee. Building Agent Models in Economic Societies of Agents. *Workshop on Agent Modelling (AAAI-96)*, 1996.

Jose M. Vidal and Edmund H. Durfee. Learning nested agent models in an information economy. *J. Exp. Theor. Artif. Intell. (UK)*, 10(3):291 – 308, 1998. ISSN 0952-813X.

Iain Wallace and Michael Rovatsos. Bounded Social Reasoning in the ESB Framework. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 1097–1104, 2009.

Iain Wallace and Michael Rovatsos. Executing specifications of social reasoning

agents. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen, editors, *Proceedings of The Eighth International Workshop on Declarative Agent Languages and Technologies*, 2010.

Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

Michael Wooldridge. *Reasoning about Rational Agents*. Intelligent robotics and autonomous agents. The MIT Press, 2000.

Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2002.

Michael Wooldridge, Marc-Phillipe Huget, Michael Fisher, and Simon Parsons. Model checking for multiagent systems: The MABLE language and its applications. *International Journal on Artificial Intelligence Tools*, 15(2):195–225, 2006.